Contents

Figures .......................... xxv
Tables .......................... xxvii
About this document .............. xxix
Who should use this document ....... xxix
How to use this document .......... xxix
z/OS information ..................... xxix
How to send your comments to IBM xxxi
If you have a technical problem ....... xxxi
Summary of changes .............. xxxiii
Summary of changes for z/OS Version 2 Release 1, as updated February 2015 xxxiii
z/OS Version 2 Release 1 summary of changes xxxiii

Chapter 1. Using the services ..... 1
Compatibility of MVS macros 1
Addressing mode (AMODE) 2
Address space control (ASC) mode 3
  ALET qualification 4
  User parameters 4
Telling the system about the execution environment 6
Specifying a macro version number 7
  How to request a macro version using PLISTVER 7
Register use 8
Handling return codes and reason codes 9
  Handling program errors 9
  Handling environmental and system errors 10
Using X-macros 11
Macro forms 11
  Conventional list form macros 12
  Alternative list form macros 12
Coding the macros 13
  Continuation lines 15
Coding the callable services 16
  Including equate (EQU) statements 17
  Link-editing linkage-assist routines 17
Service summary 17

Chapter 2. EDTINFO — Obtain eligible device table information 27
Description 27
  Environment 27
  Programming requirements 27
Restrictions 27
Input register information 28
Output register information 28
Performance implications 28
Syntax 28
Parameters 29
Return and reason codes 29
Example 30
EDTINFO - List form 30
  Syntax 30
  Parameters 31
EDTINFO - Execute form 31
  Syntax 31
  Parameters 32
EDTINFO - Modify form 32
  Syntax 32
  Parameters 33

Chapter 3. ENFREQ — Listen for system events 35
Description 35
  Environment 35
  Programming requirements 35
Restrictions 35
Input register information 36
Output register information 36
Performance implications 36
LISTEN option 36
  Syntax 36
  Parameters 38
  ENF event codes and meanings 43
  Return codes 60
Example 1 62
Example 2 63
DELETE option 64
  Syntax 64
  Parameters 64
  Return and reason codes 65
ENFREQ ACTION=LISTEN - List form 65
  Syntax 65
  Parameters 67
ENFREQ ACTION=LISTEN - Execute form 67
  Syntax 67
  Parameters 68
ENFREQ ACTION=DELETE - List form 69
  Syntax 69
  Parameters 69
ENFREQ ACTION=DELETE - Execute form 69
  Syntax 69
  Parameters 70

Chapter 4. ENQ — Request control of a serially reusable resource 71
Description 71
  Environment 71
  Programming requirements 72
Restrictions 72
Input register information 72
Output register information 73
Performance implications 73
Syntax 73
Parameters 75
ABEND codes 79
<table>
<thead>
<tr>
<th>Chapter 5. ESPIE — Extended SPIE</th>
<th>89</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>89</td>
</tr>
<tr>
<td>Environment</td>
<td>90</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>90</td>
</tr>
<tr>
<td>Restrictions</td>
<td>90</td>
</tr>
<tr>
<td>Performance implications</td>
<td>90</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>90</td>
</tr>
<tr>
<td>ESPIE SET option</td>
<td>90</td>
</tr>
<tr>
<td>Input register information</td>
<td>90</td>
</tr>
<tr>
<td>Output register information</td>
<td>90</td>
</tr>
<tr>
<td>Syntax</td>
<td>91</td>
</tr>
<tr>
<td>Parameters</td>
<td>91</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>93</td>
</tr>
<tr>
<td>Example 1</td>
<td>93</td>
</tr>
<tr>
<td>Example 2</td>
<td>93</td>
</tr>
<tr>
<td>ESPIE SET - List form</td>
<td>93</td>
</tr>
<tr>
<td>Syntax</td>
<td>93</td>
</tr>
<tr>
<td>Parameters</td>
<td>94</td>
</tr>
<tr>
<td>Example</td>
<td>94</td>
</tr>
<tr>
<td>ESPIE SET - Execute form</td>
<td>94</td>
</tr>
<tr>
<td>Syntax</td>
<td>94</td>
</tr>
<tr>
<td>Parameters</td>
<td>95</td>
</tr>
<tr>
<td>Example</td>
<td>95</td>
</tr>
<tr>
<td>ESPIE RESET option</td>
<td>95</td>
</tr>
<tr>
<td>Input register information</td>
<td>95</td>
</tr>
<tr>
<td>Output register information</td>
<td>95</td>
</tr>
<tr>
<td>Syntax</td>
<td>96</td>
</tr>
<tr>
<td>Parameters</td>
<td>96</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>96</td>
</tr>
<tr>
<td>Example</td>
<td>96</td>
</tr>
<tr>
<td>ESPIE TEST option</td>
<td>97</td>
</tr>
<tr>
<td>Input register information</td>
<td>97</td>
</tr>
<tr>
<td>Output register information</td>
<td>97</td>
</tr>
<tr>
<td>Syntax</td>
<td>97</td>
</tr>
<tr>
<td>Parameters</td>
<td>98</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>98</td>
</tr>
<tr>
<td>Example</td>
<td>98</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 6. ESTAE and ESTAEX — Specify task abnormal exit extended</th>
<th>99</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>99</td>
</tr>
<tr>
<td>Environment</td>
<td>100</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>100</td>
</tr>
<tr>
<td>Restrictions</td>
<td>100</td>
</tr>
<tr>
<td>Input register information</td>
<td>100</td>
</tr>
<tr>
<td>Output register information</td>
<td>100</td>
</tr>
<tr>
<td>Performance implications</td>
<td>101</td>
</tr>
<tr>
<td>Syntax</td>
<td>101</td>
</tr>
<tr>
<td>Parameters</td>
<td>103</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>107</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>107</td>
</tr>
<tr>
<td>Example 1</td>
<td>108</td>
</tr>
<tr>
<td>Example 2</td>
<td>108</td>
</tr>
<tr>
<td>Example 3</td>
<td>109</td>
</tr>
<tr>
<td>Example 4</td>
<td>109</td>
</tr>
<tr>
<td>Example 5</td>
<td>109</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 7. ETCON — Connect entry table</th>
<th>117</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>117</td>
</tr>
<tr>
<td>Related macros</td>
<td>117</td>
</tr>
<tr>
<td>Environment</td>
<td>117</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>117</td>
</tr>
<tr>
<td>Restrictions</td>
<td>117</td>
</tr>
<tr>
<td>Input register information</td>
<td>118</td>
</tr>
<tr>
<td>Output register information</td>
<td>118</td>
</tr>
<tr>
<td>Performance implications</td>
<td>118</td>
</tr>
<tr>
<td>Syntax</td>
<td>118</td>
</tr>
<tr>
<td>Parameters</td>
<td>119</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>120</td>
</tr>
<tr>
<td>Return codes</td>
<td>120</td>
</tr>
<tr>
<td>Examples</td>
<td>120</td>
</tr>
<tr>
<td>ETCON - List form</td>
<td>120</td>
</tr>
<tr>
<td>Syntax</td>
<td>120</td>
</tr>
<tr>
<td>Parameters</td>
<td>121</td>
</tr>
<tr>
<td>ETCON - Execute form</td>
<td>121</td>
</tr>
<tr>
<td>Syntax</td>
<td>121</td>
</tr>
<tr>
<td>Parameters</td>
<td>121</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 8. ETCRE — Create entry table</th>
<th>123</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>123</td>
</tr>
<tr>
<td>Related macros</td>
<td>123</td>
</tr>
<tr>
<td>Environment</td>
<td>123</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>123</td>
</tr>
<tr>
<td>Restrictions</td>
<td>123</td>
</tr>
<tr>
<td>Input register information</td>
<td>123</td>
</tr>
<tr>
<td>Output register information</td>
<td>124</td>
</tr>
<tr>
<td>Performance implications</td>
<td>124</td>
</tr>
<tr>
<td>Syntax</td>
<td>124</td>
</tr>
<tr>
<td>Parameters</td>
<td>125</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>125</td>
</tr>
<tr>
<td>Return codes</td>
<td>125</td>
</tr>
<tr>
<td>Example</td>
<td>125</td>
</tr>
</tbody>
</table>
### Chapter 9. ETDEF — Create an entry table descriptor (ETD)

<table>
<thead>
<tr>
<th>Description</th>
<th>127</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related macros</td>
<td>127</td>
</tr>
<tr>
<td>Environment</td>
<td>127</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>127</td>
</tr>
<tr>
<td>Restrictions</td>
<td>128</td>
</tr>
<tr>
<td>Register information</td>
<td>128</td>
</tr>
<tr>
<td>Performance implications</td>
<td>128</td>
</tr>
<tr>
<td>TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL parameters</td>
<td>128</td>
</tr>
<tr>
<td>Syntax</td>
<td>128</td>
</tr>
<tr>
<td>Parameters</td>
<td>130</td>
</tr>
<tr>
<td>TYPE=SET parameter</td>
<td>133</td>
</tr>
<tr>
<td>Syntax</td>
<td>133</td>
</tr>
<tr>
<td>Parameters</td>
<td>135</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>136</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>136</td>
</tr>
<tr>
<td>Example</td>
<td>136</td>
</tr>
</tbody>
</table>

### Chapter 10. ETDES — Destroy entry table

<table>
<thead>
<tr>
<th>Description</th>
<th>137</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related macros</td>
<td>137</td>
</tr>
<tr>
<td>Environment</td>
<td>137</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>137</td>
</tr>
<tr>
<td>Restrictions</td>
<td>137</td>
</tr>
<tr>
<td>Input register information</td>
<td>137</td>
</tr>
<tr>
<td>Output register information</td>
<td>137</td>
</tr>
<tr>
<td>Performance implications</td>
<td>137</td>
</tr>
<tr>
<td>Syntax</td>
<td>138</td>
</tr>
<tr>
<td>Parameters</td>
<td>138</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>139</td>
</tr>
<tr>
<td>Return codes</td>
<td>139</td>
</tr>
<tr>
<td>Examples</td>
<td>139</td>
</tr>
<tr>
<td>ETDES - List form</td>
<td>139</td>
</tr>
<tr>
<td>Syntax</td>
<td>139</td>
</tr>
<tr>
<td>Parameters</td>
<td>140</td>
</tr>
<tr>
<td>ETDES - Execute form</td>
<td>140</td>
</tr>
<tr>
<td>Syntax</td>
<td>140</td>
</tr>
<tr>
<td>Parameters</td>
<td>141</td>
</tr>
</tbody>
</table>

### Chapter 11. ETDIS — Disconnect entry table

<table>
<thead>
<tr>
<th>Description</th>
<th>143</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related macros</td>
<td>143</td>
</tr>
<tr>
<td>Environment</td>
<td>143</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>143</td>
</tr>
<tr>
<td>Restrictions</td>
<td>143</td>
</tr>
<tr>
<td>Input register information</td>
<td>143</td>
</tr>
<tr>
<td>Output register information</td>
<td>143</td>
</tr>
<tr>
<td>Performance implications</td>
<td>144</td>
</tr>
<tr>
<td>Syntax</td>
<td>144</td>
</tr>
<tr>
<td>Parameters</td>
<td>144</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>145</td>
</tr>
<tr>
<td>Return codes</td>
<td>145</td>
</tr>
<tr>
<td>Examples</td>
<td>145</td>
</tr>
</tbody>
</table>

### Chapter 12. EVENTS — Wait for one or more events to complete

<table>
<thead>
<tr>
<th>Description</th>
<th>147</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>147</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>147</td>
</tr>
<tr>
<td>Restrictions</td>
<td>148</td>
</tr>
<tr>
<td>Input register information</td>
<td>148</td>
</tr>
<tr>
<td>Output register information</td>
<td>148</td>
</tr>
<tr>
<td>Performance implications</td>
<td>148</td>
</tr>
<tr>
<td>Syntax</td>
<td>148</td>
</tr>
<tr>
<td>Parameters</td>
<td>149</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>150</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>150</td>
</tr>
<tr>
<td>Example 1</td>
<td>150</td>
</tr>
<tr>
<td>Example 2</td>
<td>151</td>
</tr>
</tbody>
</table>

### Chapter 13. EXTRACT — Extract TCB information

<table>
<thead>
<tr>
<th>Description</th>
<th>153</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>153</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>153</td>
</tr>
<tr>
<td>Restrictions</td>
<td>153</td>
</tr>
<tr>
<td>Performance implications</td>
<td>153</td>
</tr>
<tr>
<td>Syntax</td>
<td>154</td>
</tr>
<tr>
<td>Parameters</td>
<td>154</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>155</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>155</td>
</tr>
<tr>
<td>Example 1</td>
<td>156</td>
</tr>
<tr>
<td>Example 2</td>
<td>156</td>
</tr>
<tr>
<td>Example 3</td>
<td>156</td>
</tr>
<tr>
<td>EXTRACT - List form</td>
<td>156</td>
</tr>
<tr>
<td>Syntax</td>
<td>156</td>
</tr>
<tr>
<td>Parameters</td>
<td>157</td>
</tr>
<tr>
<td>EXTRACT - Execute form</td>
<td>157</td>
</tr>
<tr>
<td>Syntax</td>
<td>157</td>
</tr>
<tr>
<td>Parameters</td>
<td>158</td>
</tr>
</tbody>
</table>

### Chapter 14. FESTAE — Fast extended STAE

<table>
<thead>
<tr>
<th>Description</th>
<th>159</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>159</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>159</td>
</tr>
<tr>
<td>Restrictions</td>
<td>159</td>
</tr>
<tr>
<td>Input register information</td>
<td>160</td>
</tr>
<tr>
<td>Output register information</td>
<td>160</td>
</tr>
<tr>
<td>Performance implications</td>
<td>160</td>
</tr>
<tr>
<td>Syntax</td>
<td>160</td>
</tr>
<tr>
<td>Parameters</td>
<td>161</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>162</td>
</tr>
<tr>
<td>Return codes</td>
<td>162</td>
</tr>
<tr>
<td>Example</td>
<td>163</td>
</tr>
</tbody>
</table>

### Chapter 15. FRACHECK — Check user's authorization (for RACF Release 1.8.1 or earlier)

### Chapter 16. FREEMAIN — Free virtual storage

<table>
<thead>
<tr>
<th>Description</th>
<th>167</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>167</td>
</tr>
<tr>
<td>Environment</td>
<td>167</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>168</td>
</tr>
<tr>
<td>Restrictions</td>
<td>168</td>
</tr>
<tr>
<td>Input register information for SVC entry</td>
<td>168</td>
</tr>
<tr>
<td>Output register information for SVC entry</td>
<td>168</td>
</tr>
<tr>
<td>Input register information for BRANCH=YES</td>
<td>169</td>
</tr>
<tr>
<td>Output register information for BRANCH=YES</td>
<td>169</td>
</tr>
<tr>
<td>Input register information for BRANCH=(YES,GLOBAL)</td>
<td>170</td>
</tr>
<tr>
<td>Output register information for BRANCH=(YES,GLOBAL)</td>
<td>170</td>
</tr>
<tr>
<td>Performance implications</td>
<td>170</td>
</tr>
<tr>
<td>Syntax</td>
<td>170</td>
</tr>
<tr>
<td>Parameters</td>
<td>170</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>174</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>175</td>
</tr>
<tr>
<td>Example 1</td>
<td>175</td>
</tr>
<tr>
<td>Example 2</td>
<td>175</td>
</tr>
<tr>
<td>Example 3</td>
<td>175</td>
</tr>
<tr>
<td>Example 4</td>
<td>176</td>
</tr>
<tr>
<td>Example 5</td>
<td>176</td>
</tr>
<tr>
<td>FREEMAIN - List form</td>
<td>176</td>
</tr>
<tr>
<td>Parameters</td>
<td>177</td>
</tr>
<tr>
<td>FREEMAIN - Execute form</td>
<td>177</td>
</tr>
<tr>
<td>Parameters</td>
<td>178</td>
</tr>
</tbody>
</table>

Chapter 17. GETDSAB — Accessing the DSAB chain | 179
---
| Description | 179 |
| Environment | 179 |
| Programming requirements | 179 |
| Restrictions | 179 |
| Register information | 180 |
| Performance implications | 180 |
| Syntax | 180 |
| Parameters | 181 |
| Return and reason codes | 182 |
| Example 1 | 183 |
| Example 2 | 183 |
| Example 3 | 183 |
| Example 4 | 184 |
| GETDSAB - List form | 184 |
| Syntax | 184 |
| Parameters | 184 |
| GETDSAB - Execute form | 185 |
| Syntax | 185 |
| Parameters | 186 |

Chapter 18. GETMAIN — Allocate virtual storage | 187
---
| Description | 187 |
| Environment | 188 |
| Programming requirements | 189 |
| Restrictions | 189 |
| Input register information for SVC entry | 189 |
| Output register information for SVC entry | 189 |
| Input register information for BRANCH=YES | 190 |
| Output register information for BRANCH=YES | 190 |
| Input register information for BRANCH=(YES,GLOBAL) | 191 |

Chapter 19. GQSCAN — Extract information from global resource serialization queue | 205
---
| Description | 205 |
| Environment | 205 |
| Programming requirements | 205 |
| Restrictions | 206 |
| Input register information | 206 |
| Output register information | 206 |
| Performance implications | 206 |
| Syntax | 207 |
| Parameters | 207 |
| ABEND codes | 208 |
| Return and reason codes | 211 |
| Example 1 | 213 |
| Example 2 | 213 |
| Example 3 | 213 |
| Example 4 | 213 |
| Example 5 | 213 |
| GQSCAN - List form | 213 |
| Parameters | 215 |
| GQSCAN - Execute form | 215 |
| Parameters | 217 |

Chapter 20. GTRACE — GTF trace recording | 219
---
<p>| Description | 219 |
| GTRACE TEST | 219 |
| Environment | 219 |
| Programming requirements | 220 |
| Restrictions | 220 |
| Input register information | 220 |
| Output register information | 220 |
| Performance implications | 220 |
| Syntax | 220 |
| Parameters | 221 |
| ABEND codes | 221 |
| Return codes | 221 |
| GTRACE QUERY | 221 |
| Environment | 221 |
| Programming requirements | 222 |
| Restrictions | 222 |
| Input register information | 222 |
| Output register information | 222 |
| Performance implications | 222 |
| Syntax | 222 |
| Parameters | 222 |
| ABEND codes | 223 |
| Return codes | 223 |
| GTRACE DATA | 223 |
| Environment | 223 |
| Programming requirements | 223 |</p>
<table>
<thead>
<tr>
<th>Chapter 21. HISMT — HIS multithreading service</th>
<th>229</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>229</td>
</tr>
<tr>
<td>Environment</td>
<td>229</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>229</td>
</tr>
<tr>
<td>Input register information</td>
<td>230</td>
</tr>
<tr>
<td>Output register information</td>
<td>231</td>
</tr>
<tr>
<td>Performance implications</td>
<td>231</td>
</tr>
<tr>
<td>Syntax</td>
<td>231</td>
</tr>
<tr>
<td>Parameters</td>
<td>232</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>237</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>237</td>
</tr>
<tr>
<td>Example</td>
<td>239</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 22. HISSERV macro — HISSERV Service</th>
<th>243</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>243</td>
</tr>
<tr>
<td>Environment</td>
<td>244</td>
</tr>
<tr>
<td>Programming Requirements</td>
<td>244</td>
</tr>
<tr>
<td>Restrictions</td>
<td>244</td>
</tr>
<tr>
<td>Input Register Information</td>
<td>244</td>
</tr>
<tr>
<td>Output Register Information</td>
<td>245</td>
</tr>
<tr>
<td>Performance Implications</td>
<td>245</td>
</tr>
<tr>
<td>Syntax</td>
<td>245</td>
</tr>
<tr>
<td>Parameters</td>
<td>246</td>
</tr>
<tr>
<td>ABEND Codes</td>
<td>251</td>
</tr>
<tr>
<td>Return and Reason Codes</td>
<td>251</td>
</tr>
<tr>
<td>Example</td>
<td>255</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 23. HSPSERV — Read from and write to a Hiperspace</th>
<th>259</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>259</td>
</tr>
<tr>
<td>Read and write services for standard hiperspaces</td>
<td>259</td>
</tr>
<tr>
<td>Environment</td>
<td>260</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>260</td>
</tr>
<tr>
<td>Restrictions</td>
<td>260</td>
</tr>
<tr>
<td>Input register information</td>
<td>260</td>
</tr>
<tr>
<td>Output register information</td>
<td>260</td>
</tr>
<tr>
<td>Performance implications</td>
<td>261</td>
</tr>
<tr>
<td>Syntax</td>
<td>262</td>
</tr>
<tr>
<td>Parameters</td>
<td>263</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>266</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>266</td>
</tr>
<tr>
<td>Read and write services for ESO hiperspaces</td>
<td>266</td>
</tr>
<tr>
<td>Environment</td>
<td>266</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>267</td>
</tr>
<tr>
<td>Restrictions</td>
<td>267</td>
</tr>
<tr>
<td>Input register information</td>
<td>267</td>
</tr>
<tr>
<td>Output register information</td>
<td>267</td>
</tr>
<tr>
<td>Performance implications</td>
<td>267</td>
</tr>
<tr>
<td>Syntax</td>
<td>268</td>
</tr>
<tr>
<td>Parameters</td>
<td>269</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>272</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>272</td>
</tr>
<tr>
<td>HSPSERV - List form</td>
<td>273</td>
</tr>
<tr>
<td>Syntax</td>
<td>273</td>
</tr>
<tr>
<td>Parameters</td>
<td>274</td>
</tr>
<tr>
<td>HSPSERV - Execute form</td>
<td>274</td>
</tr>
<tr>
<td>Syntax</td>
<td>275</td>
</tr>
<tr>
<td>Parameters</td>
<td>276</td>
</tr>
<tr>
<td>HSPSERV - Modify form</td>
<td>276</td>
</tr>
<tr>
<td>Syntax</td>
<td>276</td>
</tr>
<tr>
<td>Parameters</td>
<td>277</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 24. IARCP64 — 64-bit cell pool services</th>
<th>279</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>279</td>
</tr>
<tr>
<td>Environment</td>
<td>279</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>280</td>
</tr>
<tr>
<td>Restrictions</td>
<td>280</td>
</tr>
<tr>
<td>Input register information</td>
<td>280</td>
</tr>
<tr>
<td>Output register information</td>
<td>281</td>
</tr>
<tr>
<td>Performance implications</td>
<td>281</td>
</tr>
<tr>
<td>Syntax</td>
<td>282</td>
</tr>
<tr>
<td>Parameters</td>
<td>284</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>293</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>293</td>
</tr>
<tr>
<td>Examples</td>
<td>294</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 25. IARR2V — Convert a central storage address to a virtual storage address</th>
<th>297</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>297</td>
</tr>
<tr>
<td>Environment</td>
<td>297</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>297</td>
</tr>
<tr>
<td>Restrictions</td>
<td>297</td>
</tr>
<tr>
<td>Input register information</td>
<td>297</td>
</tr>
<tr>
<td>Output register information</td>
<td>297</td>
</tr>
<tr>
<td>Performance implications</td>
<td>298</td>
</tr>
<tr>
<td>Syntax</td>
<td>298</td>
</tr>
<tr>
<td>Parameters</td>
<td>299</td>
</tr>
<tr>
<td>ABEND codes</td>
<td>300</td>
</tr>
<tr>
<td>Return and reason codes</td>
<td>301</td>
</tr>
<tr>
<td>Example 1</td>
<td>301</td>
</tr>
<tr>
<td>Example 2</td>
<td>302</td>
</tr>
<tr>
<td>Example 3</td>
<td>302</td>
</tr>
<tr>
<td>Example 4</td>
<td>302</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 26. IARST64 — 64-bit storage services</th>
<th>303</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>303</td>
</tr>
<tr>
<td>Environment</td>
<td>303</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>304</td>
</tr>
<tr>
<td>REQUEST=UNPROTECT option of IARV64</td>
<td>402</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>----</td>
</tr>
<tr>
<td>Environment</td>
<td>402</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>403</td>
</tr>
<tr>
<td>Restrictions</td>
<td>403</td>
</tr>
<tr>
<td>Input register information</td>
<td>403</td>
</tr>
<tr>
<td>Output register information</td>
<td>403</td>
</tr>
<tr>
<td>Performance implications</td>
<td>403</td>
</tr>
<tr>
<td>Syntax</td>
<td>404</td>
</tr>
<tr>
<td>Parameters</td>
<td>405</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQUEST=LIST option of IARV64</th>
<th>409</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>409</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>410</td>
</tr>
<tr>
<td>Restrictions</td>
<td>410</td>
</tr>
<tr>
<td>Input register information</td>
<td>410</td>
</tr>
<tr>
<td>Output register information</td>
<td>410</td>
</tr>
<tr>
<td>Performance implications</td>
<td>410</td>
</tr>
<tr>
<td>Syntax</td>
<td>410</td>
</tr>
<tr>
<td>Parameters</td>
<td>412</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQUEST=DETACH option of IARV64</th>
<th>420</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>420</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>420</td>
</tr>
<tr>
<td>Restrictions</td>
<td>420</td>
</tr>
<tr>
<td>Input register information</td>
<td>420</td>
</tr>
<tr>
<td>Output register information</td>
<td>420</td>
</tr>
<tr>
<td>Performance implications</td>
<td>421</td>
</tr>
<tr>
<td>Syntax</td>
<td>421</td>
</tr>
<tr>
<td>Parameters</td>
<td>423</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQUEST=GETSHARED option of IARV64</th>
<th>429</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>429</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>429</td>
</tr>
<tr>
<td>Restrictions</td>
<td>429</td>
</tr>
<tr>
<td>Input register information</td>
<td>429</td>
</tr>
<tr>
<td>Output register information</td>
<td>429</td>
</tr>
<tr>
<td>Performance implications</td>
<td>430</td>
</tr>
<tr>
<td>Syntax</td>
<td>430</td>
</tr>
<tr>
<td>Parameters</td>
<td>431</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQUEST=SHAREMEMOBJ option of IARV64</th>
<th>435</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>435</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>435</td>
</tr>
<tr>
<td>Restrictions</td>
<td>436</td>
</tr>
<tr>
<td>Input register information</td>
<td>436</td>
</tr>
<tr>
<td>Output register information</td>
<td>436</td>
</tr>
<tr>
<td>Performance implications</td>
<td>436</td>
</tr>
<tr>
<td>Syntax</td>
<td>436</td>
</tr>
<tr>
<td>Parameters</td>
<td>437</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQUEST=CHANGEACCESS option of IARV64</th>
<th>441</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>441</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>442</td>
</tr>
<tr>
<td>Restrictions</td>
<td>442</td>
</tr>
<tr>
<td>Input register information</td>
<td>442</td>
</tr>
<tr>
<td>Output register information</td>
<td>442</td>
</tr>
<tr>
<td>Performance implications</td>
<td>442</td>
</tr>
<tr>
<td>Syntax</td>
<td>442</td>
</tr>
<tr>
<td>Parameters</td>
<td>443</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQUEST=GETCOMMON option of IARV64</th>
<th>447</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>447</td>
</tr>
<tr>
<td>Programming requirements</td>
<td>447</td>
</tr>
<tr>
<td>Restrictions</td>
<td>448</td>
</tr>
<tr>
<td>Input register information</td>
<td>448</td>
</tr>
</tbody>
</table>

Chapter 30. IAZXCTKN — Client token compare service

Chapter 31. IAZXJSAB — Obtain information about a currently running job

Chapter 32. IEAARR — Establish an associated recovery routine (ARR)
Chapter 106. IOSVRYSW — Vary switch service ........................ 1071

Chapter 107. IOSWITCH — IOS switch information service ........ 1079

Chapter 108. IOSZHPF — zHPF channel program capabilities service 1085

Chapter 109. IQPINFO — Obtain PCIe information ..................... 1089

Chapter 110. IRDFSD — FICON switch data services ................. 1095

Chapter 111. IRDFSDU — FICON switch data update services ...... 1101

Chapter 112. ISGADMIN — Global resource serialization administration service ........................................... 1107

Chapter 113. ISGECIA — GRS enhanced contention analysis service ................................................................. 1121
## Figures

1. Sample User Parameter List for Callers in AR Mode ............... 5  
2. Sample Macro Syntax Diagram ...... 14  
3. Continuation Coding ............... 16  
4. Return Code Area Used by ENQ ............ 80  
5. Characteristics and Restrictions for Standard Hiperspaces ............ 262  
6. Characteristics and Restrictions for ESO Hiperspaces ............ 268  
7. Sample (beginning portion) Timed Event Data spreadsheet ............ 602  
8. Sample (second portion) Timed Event Data spreadsheet ............ 602  
9. RANGLIST and NUMRANGE Parameters 912
<table>
<thead>
<tr>
<th>Table Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Passing User Parameters in AR Mode</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Execution environment characteristics and corresponding SYSTSTATE parameters and global symbols</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Service Summary</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>ENF macro event codes</td>
<td>44</td>
</tr>
<tr>
<td>5</td>
<td>Return Codes for the ENFREQ Macro</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>Return Codes for the ENQ Macro with the RET=TEST Parameter</td>
<td>80</td>
</tr>
<tr>
<td>7</td>
<td>Return Codes for the ENQ Macro with the RET=USE Parameter</td>
<td>81</td>
</tr>
<tr>
<td>8</td>
<td>Return Codes for the ENQ Macro with the RET=CHNG Parameter</td>
<td>81</td>
</tr>
<tr>
<td>9</td>
<td>Return Codes for the ENQ Macro with the RET=HAVE Parameter</td>
<td>82</td>
</tr>
<tr>
<td>10</td>
<td>Return Codes for the ENQ Macro with the ECB Parameter</td>
<td>83</td>
</tr>
<tr>
<td>11</td>
<td>Return Codes for the ESPIE TEST Macro</td>
<td>98</td>
</tr>
<tr>
<td>12</td>
<td>Return and Reason Codes for the ESTAE Macro</td>
<td>107</td>
</tr>
<tr>
<td>13</td>
<td>Return and Reason Codes for the ESTAEEX Macro</td>
<td>111</td>
</tr>
<tr>
<td>14</td>
<td>Return Code for the ETCON Macro</td>
<td>120</td>
</tr>
<tr>
<td>15</td>
<td>Return Code for the ETCRE Macro</td>
<td>125</td>
</tr>
<tr>
<td>16</td>
<td>Return Codes for the ETDES Macro</td>
<td>139</td>
</tr>
<tr>
<td>17</td>
<td>Return Code for the ETDIS Macro</td>
<td>145</td>
</tr>
<tr>
<td>18</td>
<td>Return Codes for the FESTAE Macro</td>
<td>162</td>
</tr>
<tr>
<td>19</td>
<td>Return Codes for the FREEMAIN Macro</td>
<td>175</td>
</tr>
<tr>
<td>20</td>
<td>Return Codes for the GETDSAB Macro</td>
<td>182</td>
</tr>
<tr>
<td>21</td>
<td>Return and Reason Codes for the GETDSAB Macro</td>
<td>182</td>
</tr>
<tr>
<td>22</td>
<td>Return Codes for the GETMAIN Macro</td>
<td>201</td>
</tr>
<tr>
<td>23</td>
<td>Return Codes for the GQSCAN Macro</td>
<td>211</td>
</tr>
<tr>
<td>24</td>
<td>Return Codes for the GTRACE TEST Macro</td>
<td>221</td>
</tr>
<tr>
<td>25</td>
<td>Return Codes for the GTRACE DATA Macro</td>
<td>225</td>
</tr>
<tr>
<td>26</td>
<td>Return and reason codes for the HISMT macro</td>
<td>237</td>
</tr>
<tr>
<td>27</td>
<td>Return and Reason Codes for the HISSERV Macro</td>
<td>251</td>
</tr>
<tr>
<td>28</td>
<td>Return and Reason Codes for HSPSERV SREAD and HSPSERV SWRITE</td>
<td>266</td>
</tr>
<tr>
<td>29</td>
<td>Return and Reason Codes for HSPSERV CREAD and HSPSERV CWRITE</td>
<td>272</td>
</tr>
<tr>
<td>30</td>
<td>Return and Reason Codes for the IARCP64 Macro</td>
<td>293</td>
</tr>
<tr>
<td>31</td>
<td>Return and Reason Codes for the IARR2V Macro</td>
<td>301</td>
</tr>
<tr>
<td>32</td>
<td>Return and Reason Codes for the IARST64 Macro</td>
<td>316</td>
</tr>
<tr>
<td>33</td>
<td>Storage Attributes Required for Subspaces</td>
<td>323</td>
</tr>
<tr>
<td>34</td>
<td>Return and Reason Codes for the IARSUBSP Macro</td>
<td>325</td>
</tr>
<tr>
<td>35</td>
<td>IARVSERV Permitted Storage Combinations</td>
<td>330</td>
</tr>
<tr>
<td>36</td>
<td>Return and Reason Codes for the IARVSERV Macro</td>
<td>335</td>
</tr>
<tr>
<td>37</td>
<td>Return and Reason Codes for the IARV64 Macro</td>
<td>466</td>
</tr>
<tr>
<td>38</td>
<td>Return Codes for the IAZXCTKN Macro</td>
<td>472</td>
</tr>
<tr>
<td>39</td>
<td>Return and Reason Codes for the IAZXJSAB Macro</td>
<td>477</td>
</tr>
<tr>
<td>40</td>
<td>Return and reason codes for the IEALSQLRY</td>
<td>494</td>
</tr>
<tr>
<td>41</td>
<td>Return Codes for the IEAMETR Macro</td>
<td>497</td>
</tr>
<tr>
<td>42</td>
<td>Return Codes for the IEAMRMF3 Macro</td>
<td>502</td>
</tr>
<tr>
<td>43</td>
<td>Return Codes for the IEAMSCHD Macro</td>
<td>519</td>
</tr>
<tr>
<td>44</td>
<td>Return Codes for the IEANTCR Macro</td>
<td>526</td>
</tr>
<tr>
<td>45</td>
<td>Return Codes for the IEANTDL Macro</td>
<td>532</td>
</tr>
<tr>
<td>46</td>
<td>Return Codes for the IEANTRT Macro</td>
<td>536</td>
</tr>
<tr>
<td>47</td>
<td>Return Codes for the IEAN4CR Macro</td>
<td>544</td>
</tr>
<tr>
<td>48</td>
<td>Return Codes for the IEAN4DL Macro</td>
<td>550</td>
</tr>
<tr>
<td>49</td>
<td>Return Codes for the IEAN4RT Macro</td>
<td>554</td>
</tr>
<tr>
<td>50</td>
<td>Return Codes for the IEAN4RT Macro</td>
<td>554</td>
</tr>
<tr>
<td>51</td>
<td>Return and Reason Codes for the IEARBUP Macro</td>
<td>561</td>
</tr>
<tr>
<td>52</td>
<td>Return and Reason Codes for the IEATDUMP Macro</td>
<td>575</td>
</tr>
<tr>
<td>53</td>
<td>Return and reason codes for the IEATEDS macro</td>
<td>593</td>
</tr>
<tr>
<td>54</td>
<td>Return codes for the IEATXDC Macro</td>
<td>609</td>
</tr>
<tr>
<td>55</td>
<td>Authorization</td>
<td>613</td>
</tr>
<tr>
<td>56</td>
<td>Checkpoint/Restart Toleration - only available when the CVTPAUS4 bit is set in the CVT.</td>
<td>613</td>
</tr>
<tr>
<td>57</td>
<td>Authorization</td>
<td>617</td>
</tr>
<tr>
<td>58</td>
<td>Checkpoint/Restart Toleration - only available when the CVTPAUS4 bit is set in the CVT.</td>
<td>617</td>
</tr>
<tr>
<td>59</td>
<td>Linkage option</td>
<td>620</td>
</tr>
<tr>
<td>60</td>
<td>Linkage variables</td>
<td>666</td>
</tr>
<tr>
<td>61</td>
<td>Untrusted attribute linkage variable</td>
<td>666</td>
</tr>
<tr>
<td>62</td>
<td>Authorization</td>
<td>687</td>
</tr>
<tr>
<td>63</td>
<td>Checkpoint/Restart Toleration - only available when the CVTPAUS4 bit is set in the CVT.</td>
<td>687</td>
</tr>
<tr>
<td>64</td>
<td>Authorization</td>
<td>691</td>
</tr>
<tr>
<td>65</td>
<td>Checkpoint/Restart Toleration - only available when the CVTPAUS4 bit is set in the CVT.</td>
<td>691</td>
</tr>
<tr>
<td>66</td>
<td>Linkage option</td>
<td>694</td>
</tr>
<tr>
<td>67</td>
<td>Linkage variables</td>
<td>737</td>
</tr>
<tr>
<td>68</td>
<td>Untrusted attribute linkage variable</td>
<td>738</td>
</tr>
<tr>
<td>69</td>
<td>Return and Reason Codes for the IEEQEMCS Macro</td>
<td>780</td>
</tr>
<tr>
<td>70</td>
<td>Return and Reason Codes for the IEEVARYD Macro</td>
<td>790</td>
</tr>
<tr>
<td>71</td>
<td>Return Codes for the IEFPPSCN Macro</td>
<td>804</td>
</tr>
<tr>
<td>72</td>
<td>Return Codes for the IEQOMREQ Macro</td>
<td>810</td>
</tr>
<tr>
<td>73</td>
<td>Reason Codes for the IEQOMREQ Macro</td>
<td>811</td>
</tr>
<tr>
<td>74</td>
<td>Return and reason codes for the IEFSJSYM macro</td>
<td>818</td>
</tr>
<tr>
<td>75</td>
<td>Return and Reason Codes for the IEFSSI Macro</td>
<td>853</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>76</td>
<td>Return and Reason Codes for the IEFSSVT Macro</td>
<td>876</td>
</tr>
<tr>
<td>77</td>
<td>Return and Reason Codes for the IFAQUERY Macro</td>
<td>899</td>
</tr>
<tr>
<td>78</td>
<td>Parameters Valid with IOSADMF Requests</td>
<td>911</td>
</tr>
<tr>
<td>79</td>
<td>Return and Reason Codes for the IOSADMF Macro</td>
<td>913</td>
</tr>
<tr>
<td>80</td>
<td>Return and Reason Codes for the IOSCAPU CAPTUCB Macro</td>
<td>929</td>
</tr>
<tr>
<td>81</td>
<td>Return and Reason Codes for the IOSCAPU UCAPTUCB Macro</td>
<td>934</td>
</tr>
<tr>
<td>82</td>
<td>Return and Reason Codes for the IOSCAPU CAPTOACT Macro</td>
<td>939</td>
</tr>
<tr>
<td>83</td>
<td>Return and Reason Codes for the IOSCHPD Macro</td>
<td>962</td>
</tr>
<tr>
<td>84</td>
<td>Return Codes for the IOSCMB Macro</td>
<td>967</td>
</tr>
<tr>
<td>85</td>
<td>Return and Reason Codes for the IOSCUINF Macro</td>
<td>984</td>
</tr>
<tr>
<td>86</td>
<td>ABEND Codes for the IOSENQ Macro</td>
<td>997</td>
</tr>
<tr>
<td>87</td>
<td>Return and Reason Codes for the IOSENQ Macro</td>
<td>998</td>
</tr>
<tr>
<td>88</td>
<td>Return and Reason Codes for the IOSFBA Macro</td>
<td>1015</td>
</tr>
<tr>
<td>89</td>
<td>Return Codes for the IOSHXBBLK Macro</td>
<td>1021</td>
</tr>
<tr>
<td>90</td>
<td>Return Codes for the IOSLOOK Macro</td>
<td>1030</td>
</tr>
<tr>
<td>91</td>
<td>Return Codes for the IOSVRYSW Macro</td>
<td>1075</td>
</tr>
<tr>
<td>92</td>
<td>Return and reason codes for the IOSWITCH macro.</td>
<td>1083</td>
</tr>
<tr>
<td>93</td>
<td>Return codes for the IOPINFO Macro</td>
<td>1092</td>
</tr>
<tr>
<td>94</td>
<td>Return Codes for IRDFSD macro</td>
<td>1099</td>
</tr>
<tr>
<td>95</td>
<td>Return and Reason Codes for IRDFSD macro</td>
<td>1100</td>
</tr>
<tr>
<td>96</td>
<td>Return Codes for IRDFSDU macro</td>
<td>1104</td>
</tr>
<tr>
<td>97</td>
<td>Return and Reason Codes for IRDFSD macro</td>
<td>1105</td>
</tr>
<tr>
<td>98</td>
<td>Return and Reason Codes for the ISGADMIN Macro</td>
<td>1113</td>
</tr>
<tr>
<td>99</td>
<td>Return and Reason Codes for the ISGEC Macro</td>
<td>1128</td>
</tr>
<tr>
<td>100</td>
<td>Return and Reason Codes for the ISGENQ Macro</td>
<td>1151</td>
</tr>
<tr>
<td>101</td>
<td>ISGLCRT Return Codes</td>
<td>1169</td>
</tr>
<tr>
<td>102</td>
<td>ISGLID Return Codes</td>
<td>1179</td>
</tr>
<tr>
<td>103</td>
<td>ISGLOBT Return Codes</td>
<td>1185</td>
</tr>
<tr>
<td>104</td>
<td>ISGLREL Return Codes</td>
<td>1190</td>
</tr>
<tr>
<td>105</td>
<td>ISGLPRG Return Codes</td>
<td>1195</td>
</tr>
<tr>
<td>106</td>
<td>ISGLPBA Return Codes</td>
<td>1200</td>
</tr>
<tr>
<td>107</td>
<td>ISGLCR64 Return Codes</td>
<td>1204</td>
</tr>
<tr>
<td>108</td>
<td>ISGLID64 Return Codes</td>
<td>1215</td>
</tr>
<tr>
<td>109</td>
<td>ISGLOBT64 Return Codes</td>
<td>1221</td>
</tr>
<tr>
<td>110</td>
<td>ISGLRE64 Return Codes</td>
<td>1226</td>
</tr>
<tr>
<td>111</td>
<td>ISGLPRG Return Codes</td>
<td>1231</td>
</tr>
<tr>
<td>112</td>
<td>ISGLPBA64 Return Codes</td>
<td>1236</td>
</tr>
<tr>
<td>113</td>
<td>Return and Reason Codes for the ISGQUERY Macro</td>
<td>1256</td>
</tr>
<tr>
<td>114</td>
<td>Abend codes for the IITWRITE Macro</td>
<td>1276</td>
</tr>
<tr>
<td>115</td>
<td>Return and Reason Codes for the IITWRITE Macro</td>
<td>1277</td>
</tr>
<tr>
<td>116</td>
<td>Return and Reason Codes for the ITZXFILT Macro</td>
<td>1283</td>
</tr>
<tr>
<td>117</td>
<td>Return and Reason Codes for the IXGBRWSE Macro</td>
<td>1318</td>
</tr>
<tr>
<td>118</td>
<td>Return and Reason Codes for the IXGCONN Macro</td>
<td>1344</td>
</tr>
<tr>
<td>119</td>
<td>Return and Reason Codes for the IXGDELETE Macro</td>
<td>1367</td>
</tr>
<tr>
<td>120</td>
<td>Return and Reason Codes for the IXGWRITE Macro</td>
<td>1385</td>
</tr>
</tbody>
</table>
About this document

This document describes the authorized services that the MVS™ operating system provides; that is, services available only to authorized programs. An authorized program must meet one or more of the following requirements:

- Running in supervisor state
- Running under PSW key 0-7
- Running with APF-authorization.

Some of the services included in this document are not authorized, but are included because they are of greater interest to the system programmer than to the general applications programmer. The functions of these services are of such a nature that their use should be limited to programmers who write authorized programs. Services are also included if they have one or more authorized parameters — parameters available only to authorized programs.

Programmers using assembler language can use the macros described in this document to invoke the system services that they need. This document includes the detailed information — such as the function, syntax, and parameters — needed to code the macros.

This document is divided into four volumes. Volumes 1 through 4 present the macro descriptions in alphabetic order.

Who should use this document

This document is for the programmer who is using assembler language to code a system program. A system program is usually one that runs in supervisor state or runs with PSW key 0-7 or runs with APF authorization.

The document assumes a knowledge of the computer, as described in Principles of Operation, as well as an in-depth knowledge of assembler language programming.

System macros require High Level Assembler. Assembler language programming is described in the following information:

- HLASM Programmer’s Guide
- HLASM Language Reference

Using this information also requires you to be familiar with the operating system and the services that programs running under it can invoke.

How to use this document

This document is one of the set of programming documents for MVS. This set describes how to write programs in assembler language or high-level languages, such as C, FORTRAN, and COBOL. For more information about the content of this set of documents, see z/OS Information Roadmap.

z/OS information

This information explains how z/OS references information in other documents and on the web.
When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see z/OS Information Roadmap.

To find the complete z/OS® library, go to IBM Knowledge Center [http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome].
How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:
1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html).
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   US
4. Fax the comments to us, as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:
• Your name and address.
• Your email address.
• Your telephone or fax number.
• The publication title and order number:
  z/OS V2R1.0 MVS Authorized Assembler Services Reference EDT-IXG
  SA23-1373-01
• The topic and page number that is related to your comment.
• The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:
• Contact your IBM service representative.
• Call IBM technical support.
• Visit the IBM Support Portal at z/OS support page (http://www.ibm.com/systems/z/support/)
Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Summary of changes for z/OS Version 2 Release 1, as updated February 2015

The following changes are made for z/OS Version 2 Release 1 (V2R1), as updated February 2015.

New

- A new DACHMONC subtype for device monitoring has been added for ENF signal type 33 in “ENF event codes and meanings” on page 43.
- A new HISMT service has been added in Chapter 21, “HISMT — HIS multithreading service,” on page 229.
- The LOCALSYSAREA parameter is added in Chapter 24, “IARCP64 — 64-bit cell pool services,” on page 279.
- The description of return code 4, reason code 0, has been added in Chapter 113, “ISGECO — GRS enhanced contention analysis service,” on page 1121.

Changed

- Chapter 22, “HISSERV macro — HISERV Service,” on page 243 is updated for z/OS multithreading support.
- The IEAFP service is updated to modify the STOP parameter and to add the STOPVECTOR and START parameters. Return and reason codes are updated also. See Chapter 33, “IEAFP — Floating point services,” on page 485 for more information.
- Chapter 24, “IARCP64 — 64-bit cell pool services,” on page 279 is updated and restructured to improve clarity.
- The descriptions of the CSA, LSQA, RGN, and SOA options of the SDATA parameter have been updated in Chapter 45, “IEATDUMP — Transaction dump request,” on page 563.
- Information about the parameters and the return and reason codes is updated in Chapter 107, “IOSWITCH — IOS switch information service,” on page 1079.

z/OS Version 2 Release 1 summary of changes

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

- z/OS Migration
- z/OS Planning for Installation
- z/OS Summary of Message and Interface Changes
- z/OS Introduction and Release Guide
Macros and callable services are programming interfaces that application programs can use to access MVS system services. This chapter provides general information and guidelines about how to use the macros and callable services accurately and efficiently. For more specific and detailed information about coding a particular macro or callable service, see the individual service description in this information.

Some of the topics covered in this chapter apply only to macros, some apply only to callable services, and some apply to both. This chapter uses the word "services" when referring to information that applies to both service types. When information applies only to one type or the other, the particular service type is specified.

Note: z/OS macros do not code to restrictions that are imposed by the COMPAT(CASE) HLASM option or its abbreviation CPAT(CASE). Therefore, you cannot rely on using COMPAT(CASE) if you use z/OS macros.

The following table lists the topics covered in this chapter and whether the topic applies to macros, callable services, or both:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Service Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Compatibility of MVS macros”</td>
<td>Macros</td>
</tr>
<tr>
<td>“Addressing mode (AMODE)” on page 2</td>
<td>Both</td>
</tr>
<tr>
<td>“Address space control (ASC) mode” on page 3</td>
<td>Both</td>
</tr>
<tr>
<td>“ALET qualification” on page 4</td>
<td>Both</td>
</tr>
<tr>
<td>“User parameters” on page 4</td>
<td>Macros</td>
</tr>
<tr>
<td>“Telling the system about the execution environment” on page 6</td>
<td>Macros</td>
</tr>
<tr>
<td>“Specifying a macro version number” on page 7</td>
<td>Macros</td>
</tr>
<tr>
<td>“Register use” on page 8</td>
<td>Both</td>
</tr>
<tr>
<td>“Handling return codes and reason codes” on page 9</td>
<td>Both</td>
</tr>
<tr>
<td>“Handling program errors” on page 9</td>
<td>Both</td>
</tr>
<tr>
<td>“Handling environmental and system errors” on page 10</td>
<td>Both</td>
</tr>
<tr>
<td>“Using X-macros” on page 11</td>
<td>Macros</td>
</tr>
<tr>
<td>“Macro forms” on page 11</td>
<td>Macros</td>
</tr>
<tr>
<td>“Coding the macros” on page 13</td>
<td>Macros</td>
</tr>
<tr>
<td>“Coding the callable services” on page 16</td>
<td>Callable Services</td>
</tr>
<tr>
<td>“Including equate (EQU) statements” on page 17</td>
<td>Callable Services</td>
</tr>
<tr>
<td>“Link-editing linkage-assist routines” on page 17</td>
<td>Callable Services</td>
</tr>
<tr>
<td>“Service summary” on page 17</td>
<td>Both</td>
</tr>
</tbody>
</table>

### Compatibility of MVS macros

When IBM® introduces a new version or a new release of an existing version, the new version or release supports all MVS macros from previous versions and releases. Programs assembled on an earlier level of MVS that issue macros will run on later levels of MVS.

In most cases, the reverse is also true. When you assemble programs that issue macros on a particular version and release of MVS, those programs can run on earlier versions and releases of MVS, provided you request only those functions...
that are supported by the earlier version and release. This is useful for installations that write applications that might be assembled on one level of MVS, but run on a different level.

As MVS supports new architectures, addressability changes. To take best advantage of the new architectures, some macros have more than one possible expansion. You are required to have the macro expand according to the environment in which the program runs. This topic is described in this introductory information.

The problem of compatibility is not the same as selecting a macro version through the PLISTVER parameter to ensure the correct parameter list size for a macro. For selecting a parameter list version number, see "Specifying a macro version number" on page 7.

---

**Addressing mode (AMODE)**

A program can run in 24-bit, 31-bit, or 64-bit addressing mode. A program that executes in 24-bit or 31-bit addressing mode can invoke most of the services described in this information. A program that executes in 64-bit addressing mode has a smaller group of services that it can invoke.

In general,

- A program running in 24-bit addressing mode cannot pass parameters or parameter addresses that are higher than 16 megabytes. However, there are exceptions. For example, a program running in 24-bit addressing mode can:
  - Free storage above 16 megabytes using the FREEMAIN macro
  - Allocate storage above 16 megabytes using the GETMAIN macro
  - Use cell pool services for cell pools located in storage above 16 megabytes using the CPOOL macro
  - Use page services for storage locations above 16 megabytes using the PGSER macro
- A program running in 24-bit or 31-bit addressing mode cannot pass parameter addresses that are higher than 2 gigabytes, unless stated otherwise in the individual service description.
- If a program running in 31-bit or 64-bit addressing mode issues a service, parameters and parameter addresses can be above or below 16 megabytes, unless otherwise stated in the individual service description.

Some macros can generate code that is appropriate for programs in either 64-bit addressing mode or 24-bit or 31-bit addressing mode. These macros check a global symbol set by the SYSSTATE macro. See "Telling the system about the execution environment" on page 6 for more information.

When you call a callable service in 24-bit or 31-bit addressing mode, you must pass 31-bit addresses to the system service regardless of what addressing mode your program is running in. If your program is running in 24-bit mode and you use a callable service, you must set the high-order byte of parameter addresses to zeros.

You can invoke the following services in 64-bit addressing mode, subject to the “SVC or PC” restrictions mentioned later in this topic, but you cannot pass parameters and parameter addresses above 2 gigabytes: ABEND, ATTACHX, CALLDSP, CHAP, CSVQUERY, DELETE, DEQ, DETACH, DOM, DSPSERV, DYNALLOC, ENQ, ESPIE, ESTAEX, EXCP, FREEMAIN, GETMAIN, GTRACE,
IARVSERV, IDENTIFY, IEAARR, LINKXX, LOAD, MODESET, PGSER, POST, RESERVE, SDUMPX, SETRP, STAX, STIMER, STIMERM, STORAGE, SYNCHX, TIME, TIMEUSED, T TIMER, VRADATA, WAIT, WTO, WTOR, and XCTL.

There are many services that support 64-bit addressing mode and parameter addresses above 2 gigabytes. Examples are IRAV64, IARST64, and ISGENQ. For details on the supported addressing mode and parameter address ranges for any specific service, see the following books:

- z/OS MVS Programming: Assembler Services Reference ABE-HSP
- z/OS MVS Programming: Assembler Services Reference IAR-XCT
- z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN
- z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG
- z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU
- z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO
- z/OS MVS Programming: Sysplex Services Reference

Before invoking a service in 64-bit addressing mode, you must inform system macros, by specifying SYSSTATE AMODE=64, that you are in 64-bit addressing mode. You can invoke only those options that result in calling the system by an SVC or PC in 64-bit addressing mode. You cannot invoke any option that results in calling the system by a branch-entry in 64-bit addressing mode.

Unless explicitly stated otherwise, assume that a given service cannot be invoked in 64-bit addressing mode and cannot accept parameters and parameter addresses above 2 gigabytes.

For information about 64-bit addressing mode and the 64-bit GPR, see z/OS MVS Programming: Extended Addressability Guide.

**Address space control (ASC) mode**

A program can run in either primary ASC mode or access register (AR) ASC mode. In primary mode, the processor uses the contents of general purpose registers (GPRs) to resolve an address to a specific location. In AR mode, the processor uses the contents of ARs as well as the contents of GPRs to resolve an address to a specific location. See z/OS MVS Programming: Assembler Services Guide for more detailed information about AR mode.

Some macros can generate code that is appropriate for programs in either primary mode or AR mode. These macros check a global symbol set by the SYSSTATE macro. See "Telling the system about the execution environment" on page 6 for more information. Table 3 on page 18 lists the macros that check the global symbol.

Some services can generate code that is appropriate for programs in primary mode only. If you write a program in AR mode that invokes one or more services, check the description in this information for each service your program issues. Unless the description indicates that a service supports callers in AR mode, the service does not support callers in AR mode. In this case, use the SAC instruction to change the ASC mode of your program and issue the service in primary mode.

Whether the caller is in primary or AR ASC mode, the system uses ARs 0-1 and 14-15 as work registers across any service call.
ALET qualification

The address space where you can place parameters varies with the individual service:

- You can place parameters in the primary address space in all service.
- You must place parameters in the primary address space in some services.
- You can place parameters in any address space in some services.

To identify where you can locate parameters in a service, read the individual service description.

Programs in AR mode that pass parameters must use an access register and the corresponding general purpose register together (for example, access register 1 and general purpose register 1) to identify where the parameters are located. The access register must contain an access list entry token (ALET) that identifies the address space where the parameters reside. The general purpose register must identify the location of the parameters within the address space.

The only ALETs that MVS services typically accept are:

- Zero (0), which specifies that the parameters are in the caller's primary address space
- An ALET for a public entry on the caller's dispatchable unit access list (DU-AL)
- An ALET for a common area data space (CADS)

MVS services do not accept the following ALETs, and you cannot attempt to pass them to a service:

- One (1), which signifies that the parameters are in the caller's secondary address space
- An ALET that is on the caller's primary address space access list (PASN-AL) that does not represent a CADS
- An ALET for a private entry on the PASN-AL or the DU-AL

Throughout, this information uses the term AR/GPR n to mean an access register and its corresponding general purpose register. For example, to identify access register 1 and general purpose register 1, this information uses AR/GPR 1.

User parameters

Some macros that you can issue in AR mode include control parameters, user parameters, or both. Control parameters refer to the macro parameter list, and the parameters whose addresses are in the parameter list. Control parameters control the operation of the macro itself. User parameters are parameters that a user provides to be passed through to a user routine. For example, the PARAM parameter on the ATTACHX macro defines user parameters. The ATTACHX macro passes these parameters to the routine that it attaches. All other parameters on the ATTACHX macro are control parameters that control the operation of the ATTACHX macro.

Note:
1. User parameters are sometimes referred to as problem program parameters.
2. Control parameters are sometimes referred to as system parameters or control program parameters.

The macros shown in Table 1 on page 5 allow a caller in AR mode to pass information in the form of a parameter list (or parameter lists) to another routine.
This table identifies the parameter that receives the ALET-qualified address of the parameter list and tells you where the target routine finds the ALET-qualified address.

Table 1. Passing User Parameters in AR Mode

<table>
<thead>
<tr>
<th>Macro</th>
<th>Parameter</th>
<th>Location of User Parameter List Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTACH/ATTACHX</td>
<td>PARAM,VL=1</td>
<td>AR/GPR 1 contains the address of a list of addresses. When either</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• a 4-bytes-per-entry parameter list or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• an 8-bytes-per-entry parameter list with PLIST8ARALETS=YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is being used, this list also contains the ALETs associated with those addresses. (See Figure 1 for the format of the 4-bytes-per-entry parameter list when it contains ALETs.)</td>
</tr>
<tr>
<td>ESTAEX</td>
<td>PARAM</td>
<td>SDWAPARM contains the address of an 8-byte area, which contains the address and ALET of the parameter list.</td>
</tr>
</tbody>
</table>

When an AR mode caller who is using a 4-bytes-per-entry parameter list passes ALET-qualified addresses to the called program through PARAM,VL=1 on the ATTACH/ATTACHX macro, the system builds a list formatted as shown in Figure 1. The addresses passed to the called program are at the beginning of the list, and their associated ALETs follow the addresses. The last address in the list has the high-order bit on to indicate the end of the list. For example, Figure 1 shows the format of a list where an AR mode issuer of ATTACHX who is using a 4-bytes-per-entry parameter list has coded the PARAM parameter as follows:

PARAM=(A,B,C),VL=1

When an AR mode caller who is using an 8-bytes-per-entry parameter list specifies PLIST8ARALETS=YES, the system builds a parameter list with the 8-byte addresses at the beginning of the list and their associated 4-byte ALETs following the addresses.

Figure 1. Sample User Parameter List for Callers in AR Mode

For information about linkage conventions, see the chapter in z/OS MVS Programming: Assembler Services Guide.
**Telling the system about the execution environment**

To generate code that is correct for the environment in which the program runs, some macros need to know one or more of the following characteristics about that environment:

- The addressing mode (AMODE) at the time the macro is issued
- The ASC mode of the program at the time the macro is issued
- The architectural level in which the program runs

For macros that are sensitive to their environment, use the SYSSTATE macro to define the environment. During the assembly stage, SYSSTATE sets one or more global symbols. Later, in your source code, the macro checks the global symbols and generates the correct code, which might mean avoiding using a z/Architecture® instruction or an access register. Table 3 on page 18 lists MVS macros and identifies macros that need to know the environmental characteristics.

IBM recommends you issue the SYSSTATE macro before you issue other macros. Once a program has issued SYSSTATE, there is no need to reissue it, unless the program switches from one AMODE to another or one ASC mode to another or has code paths that are isolated according to architecture level or operating system release. If you switch AMODE or ASC mode to a different architecture code path, issue SYSSTATE immediately after the switch to indicate the new state. In general, issuing macros in sections of code that only run when z/OS 2.1 capabilities are available. If you do not issue the SYSSTATE macro, the system assumes the macro is issued as follows:

- In AMODE other than 64-bit
- In primary ASC mode
- Usually, in ESA/390 architectural level (but may assume z/Architecture level since all supported z/OS releases require z/Architecture level)

Table 2 describes the relevant characteristics, the corresponding parameters on the SYSSTATE macro, and the global symbols the macro checks.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Parameter on SYSSTATE</th>
<th>Global symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMODE of 64-bit, or either 24-bit or 31-bit</td>
<td>AMODE64=YES or NO</td>
<td>&amp;SYSAM64</td>
</tr>
<tr>
<td>Primary or AR ASC mode</td>
<td>ASCENV=P or AR</td>
<td>&amp;SYSASCE</td>
</tr>
<tr>
<td>Architectural level of z/Architecture</td>
<td>ARCHLVL=0, 1, 2, 3 or OSREL</td>
<td>&amp;SYSALVL</td>
</tr>
<tr>
<td>Operating system release</td>
<td>ZOSV=Rr</td>
<td>&amp;SYSOSREL</td>
</tr>
</tbody>
</table>

You can issue the SYSSTATE macro with the TEST parameter in your own user-written macro to allow your macros to generate code appropriate for their execution environment.

Callable services do not check the global symbols described in this topic. To determine whether a callable service is sensitive to the AMODE, ASC mode, or the Architecture level, see the description of the individual callable service.

In early releases of MVS, the SPLEVEL macro performs a function similar to SYSSTATE. The SPLEVEL macro identifies the level of the operating system, so that you can tune a macro expansion based on that level. You can use this where
macro expansions change incompatibly. Because SLEVEL applies to levels that the system no longer supports, it is not described in this topic.

Specifying a macro version number

Often there is more than one version of a macro, differentiated by additional parameters or new or expanded function. For example, version 1 of the IXGCONN macro provides a connection to a log stream, while version 2 adds new parameters in support of resource manager programs. This is different than using the SLEVEL macro to select a macro version level to solve problems of downward compatibility.

You can request a specific version of a macro based on the parameters you need to use in your application, but you should also be attuned to the storage constraints of the program. The version of a macro might affect the length of the parameter list generated when the macro is assembled, because when you add new parameters to a macro, the parameter list must be large enough to fit them. The size of the parameter list might grow from release to release of z/OS, perhaps affecting the amount of storage your program needs.

How to request a macro version using PLISTVER

Many macros that have one or more versions supply the PLISTVER parameter. For those that do, use the PLISTVER parameter to request a version of the macro. PLISTVER is the only parameter allowed on the list form of a macro (MF), and it determines which parameter list the system generates. PLISTVER is optional. If you omit it, the system generates a parameter list for the lowest version that will accommodate the parameters specified. This is the IMPLIED_VERSION default. Note that on the list form, the default will cause the smallest parameter list to be created.

You can also code a specific version number using plistver, or specify MAX:

- You can use plistver to code a decimal value corresponding to the version of the macro you require. The decimal value you provide determines the amount of storage allotted for the parameter list.
- You can use MAX to request that the system generate a parameter list for the highest version number currently available. The amount of storage allotted for the parameter list will depend on the level of the system on which the macro is assembled.

IBM recommends, if your program can tolerate additional growth, that you always specify PLISTVER=MAX on the list form of the macro. MAX ensures that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form when both forms are assembled using the save level of the system.

Hints for using PLISTVER

There are some general considerations that you should keep in mind when specifying the version of a macro with PLISTVER:

- If PLISTVER is omitted, the macro generates a parameter list of the lowest version that allows all the parameters specified to be processed.
- If you code PLISTVER=n and then specify any version ‘n+1’ parameter, the macro will not assemble.
- If you code PLISTVER=n and do not specify any version ‘n’ parameter, the macro will generate a version ‘n’ parameter list.
If you are using the standard form of the macro (MF=S), there is no reason you need to code the PLISTVER parameter.

Not all macros have the same version numbers. The version numbers need not be contiguous.

The PLISTVER parameter appears in the syntax diagram and in the parameter descriptions. Within each macro description, the PLISTVER parameter description specifies the range of values and lists the parameters applicable for each version of the macro.

Register use

Some services require that the caller place information in specific general purpose registers (GPRs) or access registers (ARs) prior to issuing the service. If a service has such a requirement, the “Input Register Information” topic for the service provides that information. The topic lists only those registers that have a requirement. If a register is not specified as having a requirement, then the caller does not have to place any information in that register unless using it in register notation for a particular parameter, or using it as a base register.

Once the caller issues the service, the system can change the contents of one or more registers, and leave the contents of other registers unchanged. When control returns to the caller, each register contains one of the following values or has the following status:

- The register content is preserved and is the same as it was before the service was issued.
- The register contains a value placed there by the system for the caller's use. Examples of such values are return codes and tokens.
- The system used the register as a work register. Do not assume that the register content is the same as it was before the service was issued.

Note that the system uses ARs 0, 1, 14, and 15 as work registers for every service, regardless of whether the caller is in primary or AR address space control (ASC) mode. The system does not use ARs 2 through 13 for any service.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Many macros require that the caller have a program base register and assembler USING instruction in effect when issuing the macro; that is, the caller must have program addressability. AR mode programs also require that the AR associated with the caller's base GPR be set to zero. IBM recommends the following:

- When issuing a macro, the caller should always have program addressability in effect.
- When establishing addressability, the caller should use only registers 2 through 12.

Many macros can take advantage of relative branching when they are used with the IEABRC macro or with SYSSTATE ARCHLVL=1 or SYSSTATE ARCHLVL=2, if they are running on z/OS. If relative branching is used, the caller might then need addressability only to the static data portion of the program, and not to the executable code.
Handling return codes and reason codes

Most of the services described in this information provide return codes and reason codes. Return and reason codes indicate the outcome of the service in one of the following ways:

- Successful completion: you do not need to take any action.
- Successful or partially successful completion, with additional information supplied: you should evaluate the additional information in light of your particular program and determine if you need to take any action.
- Unsuccessful completion: some type of error has occurred, and you must take some action to correct the error.

The errors that cause unsuccessful completion fall into three broad categories:

**Program errors**
- Errors that your program causes: you can correct these.

**Environmental errors**
- Errors not caused directly by your program; rather, your program's request caused a limit to be exceeded, such as a storage limit, or the limit on the size of a particular data set. You might or might not be able to correct these.

**System errors**
- Errors caused by the system: your program did nothing to cause the error, and you probably cannot correct these.

In some cases, a return or reason code can result from some combination of these errors.

The return and reason code descriptions for the services in this information indicate whether the error is a program error, an environmental error, a system error, or some combination. Whenever possible, the return and reason code descriptions give you a specific action that you can take to fix the error.

IBM recommends that you read all the return and reason codes for each service that your program issues. You can then design your program to handle as many errors as possible. When designing your program, you should allow for the possibility that future releases of MVS might add new return and reason codes to a service that your program issues.

**Handling program errors**

The actions to take in the case of program errors are usually straightforward. Typical examples of program errors are:

1. Breaking one of the rules of the service. For example:
   - Passing parameters that are either in the wrong format or not valid
   - Violating one of the environment requirements (addressing mode, locking requirements, dispatchable unit mode, and so on)
   - Providing insufficient storage for information to be returned by the system.

2. Causing errors related to the parameter list. For example:
   - Coding an incorrect combination of parameters
   - Coding one or more parameters on the service incorrectly
   - Inadvertently overlaying an area of the parameter list storage
   - Inadvertently destroying the pointer to the parameter list.
3. Requesting a service or function for which the calling program is not authorized, or which is not available on the system on which the program is running.

In each of the first two cases, you can correct your program. For completeness, the return and reason code descriptions give you specific actions to perform, even when it might seem obvious what the action should be.

In the third case, you might have to contact your system administrator or system programmer to obtain the necessary authorization, or to request that the service or function be made available on your system, and the return or reason code description asks you to take that step.

Note: Generally, the system does not take dumps for errors that your program causes when issuing a system service. If you require such a dump, then it is your responsibility to request one in your recovery routine. See the topic on providing recovery in z/OS MVS Programming: Authorized Assembler Services Guide for information about writing recovery routines.

Handling environmental and system errors

With environmental errors, often your first action should be to rerun your program or retry the request one or more times. The following are examples of environmental errors where rerunning your program or retrying the request is appropriate:

- The request being made through the service exceeds some internal system limit. Sometimes, rerunning your program or retrying the request results in successful completion. If the problem persists, it might be an indication of a larger problem requiring you to consult your system programmer, or possibly IBM support personnel. Your system programmer might be able to tune the system or cancel users so that the limit is no longer exceeded.

- The request exceeds an installation-defined limit. If the problem persists, the action might be to contact your system programmer and request that a specification in an installation exit or parmlib member be modified.

- The system cannot obtain storage, or some other resource, for your request. If the problem persists, the action might be to check with the operator to see if another user in the installation is causing the problem, or to see if the entire installation is experiencing storage constraint problems.

You might be able to design your program to anticipate certain environmental errors and handle them dynamically.

With system errors, as with environmental errors, often your first action should be to rerun your program or retry the request one or more times. If the problem persists, you might have to contact IBM support personnel.

Whenever possible for environmental and system errors, the return or reason code description gives you either a specific action you can take, or a list of recommended actions you can try.

For some errors, providing a specific action is not possible, because the action you should take depends on your particular application, and on what is happening in your installation. In those cases, the return or reason code description gives you one or more possible causes of the error to help you to determine what action to take.
Some system errors result in return and reason codes that are provided for IBM diagnostic purposes only. In these cases, the return or reason code description asks you to record the information and provide it to the appropriate IBM support personnel.

**Using X-macros**

Some MVS services support callers in both primary and AR ASC mode. When the caller is in AR mode, macros must generate larger parameter lists; the increased size of the list reflects the addition of ALETs to qualify addresses, as described under “ALET qualification” on page 4. For some MVS macros, two versions of a particular macro are available: one for callers in primary mode and one for callers in AR mode. The name of the macro for the AR mode caller is the same as the name of the macro for primary mode callers, except the AR mode macro name ends with an “X”. This information refers to these macros as **X-macros**.

The authorized X-macros are:

- ATTACHX
- ESTAEX
- SDUMFX
- SYNCHX

The only way these macros know that a caller is in AR mode is by checking the global symbol that the SYSSTATE macro sets. Each of these macros (and corresponding non-X-macro) checks the symbol. If SYSSTATE ASCENV=AR has been issued, the macro issues code that is valid for callers in AR mode. If it has not been issued, the macro generates code that is not valid for callers in AR mode. When your program returns to primary mode, use the SYSSTATE ASCENV=P macro to reset the global symbol.

**IBM recommends** that you use the X-macro regardless of whether your program is running in primary or AR mode. However, you should consider the following before deciding which macro to use:

The rules for using all X-macros, except ESTAEX, are:

- Callers in primary mode can invoke either macro.
  - Some parameters on the X-macros, however, are not valid for callers in primary mode. Some parameters on the non-X-macros are not valid for callers in AR mode. Check the macro descriptions for these exceptions.
- Callers in AR mode should issue the X-macros.
  - If a caller in AR mode issues the non-X-macro, the system substitutes the X-macro and sends a message describing the substitution.

**IBM recommends** you always use ESTAEX unless your program and your recovery routine are in 24-bit addressing mode, or your program requires a branch entry. In these cases, you should use ESTAE.

**Macro forms**

You can code most macros in three forms: standard, list, and execute. Some macros also have a modify form. When you code a macro, you use the MF parameter to select one of the forms. The list, execute and modify forms are for reenterable programs that need to change values in the parameter list of the macro. The standard form is for programs that are not reenterable, or for programs that do not change values in the parameter list.
When a program wants to change values in the parameter list of a macro, it can make the change dynamically.

However, using the standard form and changing the parameter list dynamically might cause errors. For example, after storing a new value into the inline, standard form of the parameter list, a reenterable program operating under a given task might be interrupted by the system before the program can invoke the macro. In a multiprogramming environment, another task can use the same reenterable program, and that task might change the inline parameter list again before the first task regains control. When the first task regains control, it invokes the macro. However, the inline parameter list now has the wrong values.

Through the use of the different macro forms, a program that runs in a multiprogramming environment can avoid errors related to reenterable programs. The techniques required for using the macro forms, however, are different for some macros, called alternative list form macros, than for most other macros. For the alternative list form macros, the list form description notes that different techniques are required and refers you to the information under "Alternative list form macros."

Conventional list form macros
With conventional list form macros, you can use the macro forms as follows:
1. Use the list form of the macro, which expands to the parameter list. Place the list form in the section of your program where you keep non-executable data, such as program constants. Do not code it in the instruction stream of your program.
2. In the instruction stream, code a GETMAIN or a STORAGE macro to obtain some virtual storage.
3. Code a move character instruction that moves the parameter list from its non-executable position in your program into the virtual storage area that you obtained.
4. For macros that have a modify form, you can code the modify form of the macro to change the parameter list. Use the address parameter of the modify form to reference the parameter list in the virtual storage area that you obtained. Thus, the parameter list that you change is the one in the virtual storage area obtained by the GETMAIN or STORAGE macro.
5. Invoke the macro by issuing the execute form of the macro. Use the address parameter of the execute form to reference the parameter list in the virtual storage area that you obtained.

With this technique, the parameter list is safe even if the first task is interrupted and a second task intervenes. When the program runs under the second task, it cannot access the parameter list in the virtual storage of the first task.

Alternative list form macros
Certain macros, called alternative list form macros, require a somewhat different technique for using the list form. With these macros, you do not move the area defined by the list form into virtual storage that you have obtained; instead, you place the area defined by the list form into a DSECT. Also, it is the list form, not the execute form, that you use to specify the address parameter that identifies the address of the storage for the parameter list. Note that no modify form is available for these macros.

You can use the macro forms for the alternative list form macros as follows:
1. Use the list form of the macro to define an area of storage that the execute form can use to store the parameters. As with other macros, do not code the list form in the instruction stream of your program.

2. In the instruction stream, code a GETMAIN or a STORAGE macro to obtain virtual storage for the list form expansion.

3. Place the area defined by the list form into a DSECT that maps a portion of the virtual storage you obtained.

4. Invoke the macro by issuing the execute form of the macro. The address parameter specified on the list form references the parameter list in the virtual storage area that you obtained.

**Coding the macros**

In this information, each macro description includes a syntax diagram near the beginning of the macro description. The diagram shows how to code the macro. The syntax diagram does not explain the meanings of the parameters; the meanings are explained in the parameter descriptions that follow the syntax diagram.

The syntax tables assume that the standard begin, end, and continue columns are used. Thus, column 1 is assumed as the begin column. To change the begin, end, and continue columns, use the ICTL instruction to establish the coding format you want to use. If you do not use ICTL, the assembler recognizes the standard columns. To code the ICTL instruction, see [HLASM Language Reference](#).

[Figure 2 on page 14](#) shows a sample macro, TEST, and summarizes all the coding information that is available for it. The table is divided into three columns, A, B, and C.
Column A and Column B contain those parameters that are allowed for the macro. Column A contains those parameters that are required; column B contains those parameters which are optional.

If a single line appears, as shown in A1 and B1, then that is the only available choice for the particular parameter.

If two or more lines appear together, as shown in A2 and B2, the parameters on those lines are mutually exclusive, that is, you can code any one of those parameters.

A further distinction is made between mandatory and optional parameters. The parameter descriptions that follow the syntax table clearly identify those parameters which are optional.

The third column, C, provides additional information about coding the macro.

When substitution of a variable is required in column C, the following classifications are used:

**Variable**

**Classification**

*Symbol* Any symbol valid in the assembler language. The symbol can be as long as the supported maximum length of a name entry in the assembler you are using.

*Decimal digit* Any decimal digit up to and including the value indicated in the parameter description. If both symbol and decimal digit are indicated, an absolute expression is also allowed.
Register (2)-(12)
One of general purpose registers 2 through 12, specified within parentheses, previously loaded with the right-adjusted value or address indicated in the parameter description. You must set the unused high-order bits to zero. You can designate the register symbolically or with an absolute expression.

Register (0)
General purpose register 0, previously loaded with the right-adjusted value or address indicated in the parameter description. You must set the unused high-order bits to zero. Designate the register as (0) only.

Register (1)
General purpose register 1, previously loaded with the right-adjusted value or address indicated in the parameter description. You must set the unused high-order bits to zero. Designate the register as (1) only.

Register (15)
General purpose register 15, previously loaded with the right-adjusted value or address indicated in the parameter description. You must set the unused high-order bits to zero. Designate the register as (15) only.

RX-type address
Any address that is valid in an RX-type instruction (for example, LA).

RS-type address
Any address that is valid in an RS-type instruction (for example, STM).

RS-type name
Any name that is valid in an RS-type instruction (for example, STM).

A-type address
Any address that can be written in an A-type address constant.

Default
A value that is used in default of a specified value; that is, the value the system assumes if the parameter is not coded.

Use the parameters to specify the services and options to be performed, and write them according to the following rules:
- If the selected parameter is written in all capital letters (for example, MATH, HIST, or FMT=HEX), code the parameter exactly as shown.
- If the selected parameter is written in italics (for example, grade), substitute the indicated value, address, or name.
- If the selected parameter is a combination of capital letters and italics separated by an equal sign (for example, DATA=data addr), code the capital letters and equal sign as shown, and then make the indicated substitution for the italics.
- Read the table from top to bottom.
- Code commas and parentheses exactly as shown.
- Positional parameters (parameters without equal signs) appear first; you must code them in the order shown. You may code keyword parameters (parameters with equal signs) in any order.
- If you select a parameter, read the third column before proceeding to the next parameter. The third column often contains coding restrictions for the parameter.

Continuation lines
You can continue the parameter field of a macro on one or more additional lines according to the following rules:
• Enter a continuation character (not blank, and not part of the parameter coding) in column 72 of the line.
• Continue the parameter field on the next line, starting in column 16. All columns to the left of column 16 must be blank.

You can code the parameter field being continued in one of two ways. Code the parameter field through column 71, with no blanks, and continue in column 16 of the next line; or truncate the parameter field by a comma, where a comma normally falls, with at least one blank before column 71, and then continue in column 16 of the next line. Figure 3 shows an example of each method.

Figure 3. Continuation Coding

Coding the callable services

A callable service is a programming interface that uses the CALL macro to access system services. To code a callable service, code the CALL macro followed by the name of the callable service, and a parameter list; for example:

    CALL service,(parameter list)

The syntax diagram for the sample callable service SCORE:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL SCORE</td>
<td>.(test_type, level, data, format_option, return_code)</td>
</tr>
</tbody>
</table>

Considerations for coding callable services are:
• You must code all the parameters in the parameter list because parameters are positional in a callable service interface. That is, the function of each parameter is determined by its position with respect to the other parameters in the list. Omitting a parameter, therefore, assigns the omitted parameter’s function to the next parameter in the list.
• You must place values explicitly into all input parameters, because callable services do not set default values.
• You can use the list and execute forms of the CALL macro to preserve your program’s reentrancy.
Including equate (EQU) statements

IBM supplies sets of equate (EQU) statements for use with some callable services. These statements, which you may optionally include in your source code, provide constants for use in your program. IBM provides the statements as a programming convenience to save you the trouble of coding the definitions yourself.

Note: Check the “Programming Requirements” section of the individual service description to determine if the equate statements are available for the callable service you are using. If the equate statements are available, that section will also provide a list of the statements that are provided, along with a description of how to include them in your program.

Link-editing linkage-assist routines

Linkage-assist routines provide the connection between your program and the system services that your program requests. When using callable services, link-edit the appropriate linkage-assist routines into your program module so that, during execution, the linkage-assist routines can resolve the address of, and pass control to, the requested system services. You can also dynamically link to linkage-assist routines as an alternative to link-editing. For example, issue the LOAD macro for the linkage-assist routine, then issue a CALL to the loaded addresses.

To invoke the linkage-editor or binder, code JCL as in the following example:

```
//userid JOB 'accounting-info','name','CLASS=x,
// MSGCLASS=x,NOTIFY=userid,MSGLEVEL=(1,1),REGION=4096K
//LINKSTEP EXEC PGM=HENL,
// PARM='LIST,LET,XREF,REFR,RENT'
//SYSPRINT DD SYSOUT=x
//SYSLMOD DD DSN=userid.LOADLIB,DISP=OLD
//SYSLIB DD DSN=SYS1.CSSLIB,DISP=SHR
//OBJLIB DD DSN=userid.OBJLIB,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(5,2))
//SYSLIN DD *
  INCLUDE OBJLIB(userpgm)
ENTRY userpgm
  NAME userpgm(R)
/*

Note: Omitting NCAL from the linkedit parameters (as the example shows) and specifying SYS1.CSSLIB in the //SYSLIB statement, as shown, causes the addresses of all required linkage-assist routines to be automatically resolved. This statement saves you the trouble of having to specify individual linkage-assist routines in INCLUDE statements.

Service summary

Table 3 on page 18 lists services described in the following:

- z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN
- z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG
- z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU
- z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO

For each service, the table indicates:
- Whether a program in AR ASC mode can issue the service
- Whether a program in cross memory mode can issue the service
- Whether the macro checks the SYSSTATE global macro variables
- Whether the macro can be issued in 64-bit addressing mode
Note:
1. A program running in primary ASC mode when PASN=HASN=SASN can issue any of the services listed in the table.
2. Cross memory mode means that at least one of the following conditions is true:
   \[
   \begin{align*}
   \text{PASN} \neq \text{SASN} \\
   \text{The primary address space (PASN) and the secondary address space (SASN) are different.} \\
   \text{PASN} \neq \text{HASN} \\
   \text{The primary address space (PASN) and the home address space (HASN) are different.} \\
   \text{SASN} \neq \text{HASN} \\
   \text{The secondary address space (SASN) and the home address space (HASN) are different.}
   \end{align*}
   \]
   For more information about functions that are available to programs in cross memory mode, see [z/OS MVS Programming: Extended Addressability Guide](#).
3. Callable services do not check the SYSSTATE or SPLEVEL global variables.

Table 3. Service Summary

<table>
<thead>
<tr>
<th>Service</th>
<th>Can be issued in AR ASC mode</th>
<th>Can be issued in cross memory mode</th>
<th>Checks SYSSTATE</th>
<th>Can be issued in 64-bit AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALESERV</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ASCRE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ASDES</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ASEX5</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ATSET</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ATTACH</td>
<td>Yes (See note 3 on page 25)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ATTACHX</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AXEXT</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AXFRE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AXRES</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AXREXX</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AXSET</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>BPXEKDA</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>BPXESMF</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CALLDISP</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CALLRTM</td>
<td>No</td>
<td>Yes (See note 2 on page 26)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CHANGKEY</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CIRB</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CMDAUTH</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CNZMXURF</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CNZTRKR</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>COFCREAT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>COFDEFIN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Service</td>
<td>Can be issued in AR ASC mode</td>
<td>Can be issued in cross memory mode</td>
<td>Checks SYSSTATE</td>
<td>Can be issued in 64-bit AMODE</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------</td>
<td>-----------------------------------</td>
<td>----------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>COFIDENT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>COFNOTIF</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>COFPURGE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>COFREMOV</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>COFRETRI</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>COFSDONO</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CONFCHG</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CPF</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CPOOL</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CPUTIMER</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CSRSI</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CSRUNIC</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CSVAPF</td>
<td>Yes(^{(1)}) (See note 11 on page 26)</td>
<td>Yes(^{(12)}) (See note 12 on page 26)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CSVDYNEX</td>
<td>Yes(^{(13)}) (See note 13 on page 26)</td>
<td>Yes(^{(14)}) (See note 14 on page 26)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CTRACE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CTRACECS</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CTRACEWR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DATOFF</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DEQ</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DIV</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DOM</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DSPSERV</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DYNALLOC</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>EDTINFO</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ENFREQ</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ENQ</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ESPIE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ESTAE (See note 3 on page 26)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ESTAEX</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ETCON</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ETCRE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ETDEF</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ETDES</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ETDIS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>EVENTS</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>EXTRACT</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 3. Service Summary (continued)

<table>
<thead>
<tr>
<th>Service</th>
<th>Can be issued in AR ASC mode</th>
<th>Can be issued in cross memory mode</th>
<th>Checks SYSSTATE</th>
<th>Can be issued in 64-bit AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FESTAE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>FREEMAIN</td>
<td>Yes (See note [4] on page 26)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GETDSAB</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>GETMAIN</td>
<td>Yes (See note [4] on page 26)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GQSCAN</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>GTRACE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>HSPSERV</td>
<td>Yes</td>
<td>Yes (See note [5] on page 26)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IARCP64</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IARR2V</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IARSUBSP</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IARST64</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IARVSERV</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IARV64</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IAZXCTKN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IAZXJSAB</td>
<td>Yes</td>
<td>Yes (See note [15] on page 26)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEAARR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IEAFP</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEALSQRY</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEAMETR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEAMRMF3</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEAMSCHD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEANTCR</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>IEANTDL</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>IEANTRT</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>IEARBUP</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEATDUMP</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEATEDS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEATXDC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IEAVAPE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEAVAPE2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEAVDPE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEAVDPE2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEAVPSE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEAVPSE2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEAVRLS</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEAVRLS2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
### Table 3. Service Summary (continued)

<table>
<thead>
<tr>
<th>Service</th>
<th>Can be issued in AR ASC mode</th>
<th>Can be issued in cross memory mode</th>
<th>Checks SYSSTATE</th>
<th>Can be issued in 64-bit AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAVRPI</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEAVRPI2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEAVTPE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEAVXFR</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEAVXFR2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEA4APE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4APE2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4DPE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4DPE2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4PSE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4PSE2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4RLS</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4RLS2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4RPI</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4RPI2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4TPE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4XFR</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4XFR2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEECMDS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEEQEMCS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEEVARYD</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEFPPSCN</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEFQMREQ</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEFSSI</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEFSSVT</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEFSSVTI</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IFAQUERY</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IOCINFO</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IOSADMF</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IOSCAPF</td>
<td>No</td>
<td>Yes</td>
<td>(See note 7 on page 26)</td>
<td>No</td>
</tr>
<tr>
<td>IOSCAPU</td>
<td>Yes</td>
<td>Yes</td>
<td>(See note 7 on page 26)</td>
<td>No</td>
</tr>
<tr>
<td>IOSCDR</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IOSCHPD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IOSCMXA</td>
<td>No</td>
<td>Yes</td>
<td>(See note 7 on page 26)</td>
<td>No</td>
</tr>
<tr>
<td>IOSCMXR</td>
<td>No</td>
<td>Yes</td>
<td>(See note 7 on page 26)</td>
<td>No</td>
</tr>
<tr>
<td>Service</td>
<td>Can be issued in AR ASC mode</td>
<td>Can be issued in cross memory mode</td>
<td>Checks SYSSTATE</td>
<td>Can be issued in 64-bit AMODE</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------</td>
<td>-----------------------------------</td>
<td>----------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>IOSDCXR</td>
<td>No</td>
<td>Yes (See note <a href="#">7</a>)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IOSENQ</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IOSINFO</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IOSLOOK</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IOSPTHV</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IOSSPOF</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IOSUPFA</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IOSUPFR</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IOSVRYSW</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IOSWITCH</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IOSZHPF</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IRDFS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IRDFSU</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ISGADMIN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ISGECA</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ISGENQ</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ISGLCRT (See note <a href="#">16 on page 26</a>)</td>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>ISGLID (See note <a href="#">16 on page 26</a>)</td>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>ISGLOBT</td>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>ISGLREL</td>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>ISGLPRG</td>
<td>No</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>ISQUERY</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ITTFMTB</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ITZXFILT</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMCLSFY</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IWMCONN</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMDISC</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMECQRY</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMECREA</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMEDLE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMMABNL</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IWMMCHEST</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IWMMCREA</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMMEDLE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMMEXTR</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 3. Service Summary (continued)

<table>
<thead>
<tr>
<th>Service</th>
<th>Can be issued in AR ASC mode</th>
<th>Can be issued in cross memory mode</th>
<th>Checks SYSSTATE</th>
<th>Can be issued in 64-bit AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWMMINIT</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IWMMENTF</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMMEMREL</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMMSWCH</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMMXFER</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IWMPQRY</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMRCOLL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMRPT</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMPQRY</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMSRDRS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMSRSRG</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMSRSRS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMWMCON</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMWQRY</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IWMWQWRK</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXCCREAT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXCDELETE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXJOIN</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXCLEAVE</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXCMG</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXCMOD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXMSGI</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXMSGO</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXQUERY</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXQUIES</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXSETUS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXTERM</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXGBRWSE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DXGCN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DXGDELET</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DXGWRITE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LLACOPY</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>LOAD</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>LOADWAIT</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>LOCASCB</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>LXFRE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>LXRES</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MCSOPER</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Chapter 1. Using the services 23
<table>
<thead>
<tr>
<th>Service</th>
<th>Can be issued in AR ASC mode</th>
<th>Can be issued in cross memory mode</th>
<th>Checks SYSSTATE</th>
<th>Can be issued in 64-bit AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCSOPMSG</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MGCR</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MGCRE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MIQUERY</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MODESET</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>NIL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>NMLDEF</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NUCLKUP</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>OIL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>OUTADD</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>OUTDEL</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PCLINK</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PGANY</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PGFIX</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>PGFIXA</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PGFREE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PGFREEA</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PGSER</td>
<td>Yes (See note [5] on page 26)</td>
<td>Yes (See note [5] on page 26)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>POST</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>PTRACE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PURGEDQ</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>QEDIT</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RESERVE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RESMGR</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RESUME</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RISGNL</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SCEDIRB</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SCHEDULE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SCHEDXIT</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SDUMP</td>
<td>Yes (See note [1] on page 25)</td>
<td>Yes (See note [2] on page 26)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SDUMPX</td>
<td>Yes</td>
<td>Yes (See note [2] on page 26)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SETFRR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SETLOCK</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SETRP</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SJFREQ</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SPIE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
### Table 3. Service Summary (continued)

<table>
<thead>
<tr>
<th>Service</th>
<th>Can be issued in AR ASC mode</th>
<th>Can be issued in cross memory mode</th>
<th>Checks SYSSTATE</th>
<th>Can be issued in 64-bit AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPOST</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SRBSTAT</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SRBTIMER</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>STATUS</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>STORAGE</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SUSPEND</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SVCUPDTE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SWAREQ</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SWBTRREQ</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SYMREC</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SYNCH</td>
<td>Yes (See note 1)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SYNCHX</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSEVENT</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TCBTOKEN</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TCTL</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TESTAUTH</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TIMEUSED</td>
<td>Yes (See note 10 on page 26)</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>T6EXIT</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>UCBINFO</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>UCBLOOK</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>UCBPIN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>UCBSCAN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>VSMLIST</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>VSMLOC</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>VSMREGN</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>WAIT</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>WTL</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>WTO</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>WTOR</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Notes:**

1. Primary mode callers can use either macro in the following macro pairs:
   - ATTACH or ATTACHX
   - SDUMP or SDUMPX
   - SYNCH or SYNCHX

**IBM recommends** that programs in AR ASC mode use the X-macros (ATTACHX, SDUMPX, and SYNCHX). If, however, a program in AR mode issues ATTACH, SDUMP, or SYNCH after issuing SYSSTATE ASCENV=AR, the system substitutes the corresponding X-macro and issues a message telling you that it made the substitution.
2. CALLRTM TYPE=MEMTERM can be issued in cross memory mode. For CALLRTM TYPE=ABTERM, see the CALLRTM macro description.

3. The only programs that can use ESTAE are programs that are in primary mode with (PASN=HASN=SASN).

   **IBM recommends** you always use ESTAEX unless your program and your recovery routine are in 24-bit addressing mode, or your program requires a branch entry. In these cases, you should use ESTAE.

4. IBM recommends that AR mode callers use the STORAGE macro instead of using GETMAIN or FREEMAIN.

5. For HSPSERV SREAD and HSPSERV SWRITE, PASN=HASN=SASN for a non-shared standard hiperspace for which an ALET is not used (that is, the HSPALET parameter is omitted).

6. If you use the HSPALET parameter, the HSPSERV macro checks SYSSTATE.

7. If the input UCB is captured, the IOSCAPF, IOSCMXA, IOSCMXR, and IOSDCXR macros can be issued in cross memory mode only if the UCB is captured in the primary address space. IOSCAPU CAPTOACT without the ASID parameter also can be issued in cross memory mode if the UCB was captured in the primary address space. IOSCAPU CAPTUCB and IOSCAPU UCAPTUCB cannot be issued in cross memory mode.

8. PGSER can be issued in AR ASC mode only if you specify BRANCH=Y. PGSER can be issued in cross memory mode only if you specify BRANCH=Y or BRANCH=SPECIAL.

9. Both SDUMP and SDUMPX can be issued in cross memory mode only if you specify BRANCH=YES.

10. Only TIMEUSED LINKAGE=SYSTEM can be issued in AR ASC mode. TIMEUSED LINKAGE=BRANCH cannot be issued in AR ASC mode.

11. For a QUERY request, CSVAPF can be issued only in primary mode. For all other requests, CSVAPF can be issued in primary or AR mode.

12. For CSVAPF with the ADD, DELETE, and DYNFORMAT requests, PASN = HASN = SASN. For CSVAPF with the QUERY, QUERYFORMAT, and LIST requests, any PASN, any HASN, any SASN.

13. For a QUERY or a CALL request with FASTPATH=YES, CSVDYNEX can be issued only in primary mode. For all other requests, CSVDYNEX can be issued in primary or AR mode.

14. For CSVDYNEX CALL, RECOVER, and QUERY requests, any PASN, any HASN, any SASN. For all other requests, PASN=HASN=SASN.

15. When the caller of the IAZXJSAB macro specifies the ASCB parameter, any PASN, any HASN, any SASN; otherwise, PASN=HASN is required.

16. The 64 bit entry names are as follows:
   - ISGLCR64
   - ISGLID64
   - ISGLOB64
   - ISGLRE64
   - ISGLPB64
   - ISGLPR64
Chapter 2. EDTINFO — Obtain eligible device table information

Description

The EDTINFO macro enables you to obtain information from the eligible device table (EDT) and to check your device specification against the information in the EDT. See z/OS HCD Planning and z/OS MVS Programming: Assembler Services Guide for further information about the EDT.

For callers only in supervisor state AND PSW key 0, the EDTINFO macro performs the following function:

- Return EDT Latch Tables (RTNEDTLT)

Note: If the RTNEDTLT function is specified, no other EDTINFO functions can be requested on the same invocation nor can the IOCTOKEN and EDTADDR keywords be specified.

For a list of functions performed by the EDTINFO macro for both unauthorized and authorized callers, see z/OS MVS Programming: Assembler Services Reference ABE-HSP.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state and any PSW key 0.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN=SASN or PASN==HASN==SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24- or 31- bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

Callers requesting the RTNEDTLT function of the EDTINFO macro must be in 31-bit AMODE to reference the areas returned through the ELTPRI and ELTSEC pointers.

Callers requesting the RTNEDTLT function of the EDTINFO macro are required to free the storage returned through the ELTPRI and ELTSEC pointers.

Restrictions

Callers must be supervisor state and PSW key 0 in order to invoke the RTNEDTLT function.
**EDTINFO macro**

**Input register information**
Before issuing the EDTINFO macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

**Output register information**
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR 15 contains a return code of 04 or 08; otherwise, used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**
None.

**Syntax**
The standard form of the EDTINFO macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede EDTINFO.</td>
</tr>
<tr>
<td>EDTINFO</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow EDTINFO.</td>
</tr>
<tr>
<td>RTNEDTLT</td>
<td>Note: If this function is specified, no other functions can be requested.</td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

RTNEDTLT
   Specifies that the EDTINFO service should return both the primary and secondary EDT Latch Tables.

,ELTPRI=eltpri
   Specifies the fullword output field that will contain the address of the primary EDT Latch Table. The area returned can be mapped by IEFDELT.

ELTSEC=eltsec
   Specifies the fullword output field that will contain the address of the secondary EDT Latch Table. The area returned can be mapped by IEFDELT.

,RETCODE=retcode
   Specifies the fullword location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=rsncode
   Specifies the fullword location where the system is to store the reason code. The reason code is also in GPR 0.

Return and reason codes

When control returns from EDTINFO, GPR 15 (and retcode addr, if you coded RETCODE) contains one of the following hexadecimal return codes:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>The requested function or functions were performed and no reason code information has been returned.</td>
</tr>
<tr>
<td>04</td>
<td>The requested function or functions were performed and information has been returned, as explained by the hexadecimal reason code that accompanies this return code. The reason code is in GPR 0 (and in rsncode, if you coded RSNCODE).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>Either the primary EDT Latch Table or the secondary EDT Latch Table or both EDT Latch Tables contain no entries.</td>
</tr>
</tbody>
</table>
Return Code | Meaning
---|---
08 | There is data in the input parameter list that is not valid, as explained by the hexadecimal reason code that accompanies this return code. The reason code is in GPR 0 (and in rsncode, if you coded RSNCODE).

### Reason Code

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
</tr>
<tr>
<td>02</td>
</tr>
<tr>
<td>03</td>
</tr>
<tr>
<td>04</td>
</tr>
<tr>
<td>05</td>
</tr>
<tr>
<td>06</td>
</tr>
<tr>
<td>07</td>
</tr>
<tr>
<td>08</td>
</tr>
<tr>
<td>09</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>18</td>
</tr>
</tbody>
</table>

### Example

Obtain the EDT Latch Tables for both the primary and secondary EDTs.

```assembler
EDTINFO RTNEDTLT,ELTPRI=PRI_ELT_PTR,ELTSEC=SEC_ELT_PTR
```

### EDTINFO - List form

Use the list form of the EDTINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses for storing the parameters.

#### Syntax

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros.

The list form of the EDTINFO macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede EDTINFO.</td>
</tr>
<tr>
<td>EDTINFO</td>
<td></td>
</tr>
</tbody>
</table>
EDTINFO macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/bslash</td>
<td>One or more blanks must follow EDTINFO.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MF=(L,list addr)</th>
<th>list addr: Symbol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF=(L,list addr,attr)</td>
<td>attr: 1- to 60-character input string</td>
</tr>
<tr>
<td>MF=(L,list addr,0D)</td>
<td>Default: 0D</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

- **MF=(L,list addr)**
- **MF=(L,list addr,attr)**
- **MF=(L,list addr,0D)**

Specifies the list form of the EDTINFO macro.

The list addr parameter specifies the address of the storage area for the parameter list.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

EDTINFO - Execute form

Use the execute form of the EDTINFO macro together with the list form for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the EDTINFO macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/bslash</td>
<td>One or more blanks must precede EDTINFO.</td>
</tr>
<tr>
<td>RTNEDTLT</td>
<td>Note: If this function is specified, no other functions can be requested.</td>
</tr>
<tr>
<td>,ELTPRI=eltpri</td>
<td>eltpri: RS-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
EDTINFO macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,ELTSEC=eltsec</td>
<td>eltsec: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>MF=(E,list addr)</td>
<td>list addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>MF=(E,list addr,COMPLETE)</td>
<td>Default: COMPLETE</td>
</tr>
<tr>
<td>MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained under the standard form of the EDTINFO macro with the following exceptions:

\,MF=(E,list addr)
\,MF=(E,list addr,COMPLETE)
\,MF=(E,list addr,NOCHECK)

Specifies the execute form of the EDTINFO macro.

The list addr parameter specifies the address of the storage area for the parameter list.

COMPLETE specifies that the system is to check for required parameters and supply defaults for optional parameters that were not specified. NOCHECK specifies that the system does not check for required parameters and does not supply defaults for optional parameters that were not specified.

Note: When using the NOCHECK option, make sure that it is preceded by an execute or modify form invocation that specifies or defaults to the COMPLETE option. Otherwise, the parameter list might not be completely initialized.

EDTINFO - Modify form

Use the modify form of the EDTINFO macro to change parameters in the control parameter list that the system created through the list form of the macro.

Syntax

The modify form of the EDTINFO macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede EDTINFO.</td>
</tr>
<tr>
<td>EDTINFO</td>
<td></td>
</tr>
</tbody>
</table>
### EDTINFO macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>One or more blanks must follow EDTINFO.</td>
</tr>
<tr>
<td>RTNEDTLT</td>
<td><strong>Note</strong>: If this function is specified, no other functions can be requested.</td>
</tr>
</tbody>
</table>

- **,ELTPRI=eltpri**
  
  `eltpri`: RS-type address or register (2) - (12).

- **,ELTSEC=eltsec**
  
  `eltsec`: RS-type address or register (2) - (12).

- **,RETCODE=retcode**
  
  `retcode`: RX-type address or register (2) - (12).

- **,RSNCODE=rsncode**
  
  `rsncode`: RX-type address or register (2) - (12).

- **,MF=(M,list addr)**
  
  `list addr`: RX-type address or register (2) - (12).

- **,MF=(M,list addr,COMPLETE)**
  
  Default: COMPLETE

- **,MF=(M,list addr,NOCHECK)**

### Parameters

The parameters are explained under the standard form of the EDTINFO macro with the following exceptions:

- **,MF=(M,list addr)**

- **,MF=(M,list addr,COMPLETE)**

- **,MF=(M,list addr,NOCHECK)**

  Specifies the modify form of the EDTINFO macro.

  The `list addr` parameter specifies the address of the storage area for the parameter list.

  COMPLETE specifies that the system is to check for required parameters and supply defaults for optional parameters that were not specified. NOCHECK specifies that the system does not check for required parameters and does not supply defaults for optional parameters that were not specified.

**Note**: When using the NOCHECK option, make sure that it is preceded by an execute or modify form invocation that specifies or defaults to the COMPLETE option. Otherwise, the parameter list might not be completely initialized.
EDTINFO macro
Chapter 3. ENFREQ — Listen for system events

Description

The ENFREQ macro enables an authorized program to:

- Register to be notified when an ENF-defined event occurs (ACTION=LISTEN),
- Delete registration for notification of an ENF-defined event (ACTION=DELETE).

To listen for an event, a program issues ENFREQ with the ACTION=LISTEN parameter. When the event that the program is listening for occurs, control passes to the listener user exit routine specified on the EXIT or SRBEXIT parameter. For a list of the events for which a program can listen, see Table 4 on page 44.

To stop listening for an event, a program issues ENFREQ with the ACTION=DELETE parameter to delete the listen request. When a program issues ENFREQ with the ACTION=DELETE parameter, ENF either deletes the listen request immediately if the listener user exit has completed, or waits until the listener user exit completes. Because the listener user exit might not have completed processing at the time the delete request is issued, you must not release the listener user exit's storage or any resources that may be required by the exit. ENF does not delete the user exit when it deletes a listen request. See “DELETE option” on page 64 for the syntax of a delete request.

For guidance information about how to use the ENFREQ macro and code the listener user exit routine, see z/OS MVS Programming: Authorized Assembler Services Guide.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state and any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external</td>
</tr>
<tr>
<td></td>
<td>interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address</td>
</tr>
<tr>
<td></td>
<td>space</td>
</tr>
</tbody>
</table>

Programming requirements

The caller of ENFREQ must do the following:

- Include the CVT, IEFENFCT, and IEFENFPM mapping macros. Specify the DSECT=YES option with the CVT mapping macro.
- Declare a fullword and label it ENFPTR.

Restrictions

None.
ENFREQ macro

Input register information
Before issuing the ENFREQ macro, the caller must ensure that the following GPRs contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 18-word save area.</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller of the ENFREQ macro, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unchanged</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller of the ENFREQ macro, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Reason code</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

LISTEN option

Syntax
The standard form of the ENFREQ macro for ACTION=LISTEN is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
</tbody>
</table>
## ENFREQ macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>One or more blanks must precede ENFREQ.</td>
</tr>
</tbody>
</table>

**ENFREQ**

| b | One or more blanks must follow ENFREQ. |

**ACTION=LISTEN**

| ,CODE=**event code** | *event code*: Decimal digit or symbol. |
| ,DTOKEN=**dtoken** | *dtoken*: RX-type address or address in register (2) - (12). |

| ,DISABLE=NO | **Default**: DISABLE=YES |
| ,DISABLE=YES | |

| ,ESTBNME=**estab name** | *estab name*: RX-type address or address in register (2) - (12). |
| ,EXITNME=**exitrtn name** | *exitrtn name*: RX-type address or address in register (2) - (12). |

| ,MASEXIT=No | **Default**: MASEXIT=NO |
| ,MASEXIT=YES | |

| ,FLTRBLK=**filter block addr** | *filter block addr*: RX-type address or address in register (2) – (12). |
| ,QUAL=**qualifier** | *qualifier*: RX-type address or address in register (2) - (12). |

| | **Default**: QMASK=NONE |

| ,BITQUAL=**bitqual** | *bitqual*: name of a 32-byte field, hexadecimal numeric value (X‘xxx’), or address in register (2) - (12) |

| | **Default**: 32 bytes of X’00’. |

| ,BITCOMPARE=SUBSET | **Default**: BITCOMPARE=SUBSET |
| ,BITCOMPARE=INTERSECT | |
| ,BITCOMPARE=EQUAL | |

| ,SRBEXIT=**exitrtn addr** | *exitrtn addr*: A-type address, or address in register (2) - (12). |
| ,EXIT=**exitrtn addr** | *exitrtn addr*: A-type address or address in register (2) - (12). |

| ,PARM=**parm addr** | *parm addr*: A-type address, or address in register (2) - (12). |
| ,PARM=**parm data** | *parm data*: a fullword of data |
### ENFREQ macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,EOT=NO</td>
<td>Default: EOT=NO.</td>
</tr>
<tr>
<td>,EOT=YES</td>
<td></td>
</tr>
<tr>
<td>,EOM=NO</td>
<td>Default: EOM=NO.</td>
</tr>
<tr>
<td>,EOM=YES</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=2</td>
<td>Default: Version implied by keywords</td>
</tr>
<tr>
<td>,PLISTVER=3</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,RELATED=(value)</td>
<td>value: Any text.</td>
</tr>
<tr>
<td>,XSYS=NO</td>
<td>Default: XSYS=NO.</td>
</tr>
<tr>
<td>,XSYS=YES</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

- **ACTION=LISTEN**
  - A required parameter that specifies that you want to listen for a specific system event.

- **CODE=** *event code*
  - A required parameter that specifies the system event about which the caller wants to be notified. The *event code* can be any of the decimal codes listed in Table 4 on page 44.

- **DTOKEN=** *dtoken*
  - Specifies a 4-byte output field into which the event notification facility (ENF) returns a token to identify the request. To explicitly delete the listen request in the future, you must code this parameter.

- **DISABLE=NO**
- **DISABLE=YES**
  - Indicates if the listen exit should be disabled for future calls in the case where ENF enters into recovery processing because of an abend or other error in the listen exit. The default is DISABLE=YES.

- **ESTBNME=** *estab name*
  - Specifies the name of the establisher of the listener user exit routine. The name can be 1 to 8 alphanumeric characters. This optional parameter can be helpful for diagnostic purposes. If you specify ESTBNME, you must also specify EXITNME.

- **EXITNME=** *exitrtn name*
  - Specifies the name of the listener user exit routine to receive control when the requested event occurs. The name can be 1 to 8 alphanumeric characters. This optional parameter can be helpful for diagnostic purposes. If you specify EXITNME, you must also specify ESTBNME.
Specifies whether multiple address spaces can use the same listen exit. MASEXIT=NO, the default, specifies that only one address space can use a particular listen exit. MASEXIT=YES specifies that other address spaces can use the same listen exit.

**Note:** If you specify the MASEXIT parameter, you cannot also specify the SRBEXIT parameter.

Specifies the address of an ENF Listener Filter block. This can only be specified for ENF codes that support it. See Table 4 on page 44 for the ENF codes that support filter blocks. The filter data is specific to the signal code. See the ENF Codes And Meanings section of the z/OS MVS Programming: Authorized Assembler Services Guide to find the macro mapping name which would contain the filter block data. The storage containing the filter block data can be released following the ENFREQ LISTEN request.

Specifies a four-byte value. The four-byte value, called a qualifier, further defines the event. The qualifiers that are valid depend on the system event for which you are listening. Table 4 on page 44 lists the meaning of the valid QUAL values for each event.

To specify the bytes of the qualifier that ENF is to use, code any combination of the following keywords separated by commas.

- **BYTE1**
  - First byte
- **BYTE2**
  - Second byte
ENFREQ macro

BYTE3
Third byte

BYTE4
Fourth byte

ALL  All four bytes

NONE  No bytes

\BITQUAL=bitqual
Specifies a 32-byte field, a hexadecimal constant, or a register containing the
description of a 32-byte field containing a bit-mapped qualifier that further defines
the event. The qualifiers that are valid depend on the system event for which
you are listening.

To use this keyword, set BITQUAL as described in Table 4 on page 44. The
figure also lists the mapping macro that defines symbolic values for the
qualifier, if any. If you do not specify BITQUAL, the system responds as if you
had provided a bit-mapped qualifier with all bits set to zero.

The listen exit receives control only when a system event occurs that matches
the characteristics specified by the bit-mapped qualifier and the comparison
operation specified by the BITCOMPARE parameter. For example, if
BITCOMPARE=INTERSECT, the listener user exit receives control when an
event with characteristics represented by any of the bits that are set to ‘1’ in
the bit-mapped qualifier occurs.

If your listen request also specifies the QUAL keyword, the listen exit receives
control only when the system event also matches the characteristics specified
by the QMASK bytes of the QUAL field. The system event is only delivered if
your listen request also specifies the FLTRBLK keyword. The listen exit
receives control only when those filters are also passed. See the description of
the QUAL and QMASK keywords in this information.

\BITCOMPARE=SUBSET
\BITCOMPARE=INTERSECT
\BITCOMPARE=EQUAL

Specifies the comparison operation ENF uses to interpret the bit-mapped
qualifier specified with the BITQUAL parameter. In the examples provided
with the following parameter descriptions, only 8 of the 256 bits in the
bit-mapped qualifier are shown.

• SUBSET, the default, specifies that ENF is to pass control to the listener user
  exit when an event with characteristics represented by all of the bits that are
  set to ‘1’ in the bit-mapped qualifier occurs.

  For example, if BITQUAL=X'A0...' (B'10100000...') and
  BITCOMPARE=SUBSET, ENF will pass control to the listener user exit for a
  system event described by any of the following bit patterns:
  – B'10100000...'
  – B'11100000...'
  – B'10111111...

  Note: The above list is not exhaustive.

  In all these cases, the characteristics described by the BITQUAL parameter
  are a subset of the event's characteristics. That is, every bit set to ‘1’ in the
  bit-mapped qualifier is also set to ‘1’ in the bit pattern describing the system
  event.
ENFREQ macro

- **INTERSECT** specifies that ENF is to pass control to the listener user exit when an event with characteristics represented by any of the bits that are set to ‘1’ in the bit-mapped qualifier occurs.

  For example, if BITQUAL=X’S0...’ (B’10100000...’) and BITCOMPARE=INTERSECT, ENF will pass control to the listener user exit for a system event described by any of the following bit patterns:
  - B’10000000...’
  - B’00100000...’
  - B’10111111...’

  **Note:** This list is not exhaustive.

  In all these cases, the intersection of the characteristics described by the BITQUAL parameter and the characteristics of the event is non-null. At least one bit set to ‘1’ in the bit-mapped qualifier is also set to ‘1’ in the bit pattern describing the system event.

- **EQUAL** specifies that ENF is to pass control to the listener user exit when an event with characteristics exactly represented by the bit-mapped qualifier occurs.

  For example, if BITQUAL=X’S0...’ (B’10100000...’) and BITCOMPARE=EQUAL, ENF will pass control to the listener user exit only for a system event described by the bit pattern B’10100000...’.

  In this case, the characteristics described by the BITQUAL parameter exactly match the characteristics of the system event that has occurred, and the bit-mapped qualifier exactly matches the bit pattern describing the system event.

To specify that ENF is not to consider the bit-mapped qualifier when determining whether the listener user exit is to receive control, do one of the following:

- Omit both the BITQUAL and the BITCOMPARE parameters, or
- Specify BITQUAL=0 and BITCOMPARE=SUBSET

\_SRBEXIT=exitrtn addr

Specifies the address of a listener user exit routine that receives control when the requested event occurs. The specified routine receives control in SRB mode in the address space that issued the listen request. SRBEXIT is valid only with certain event codes. The combination of EOM=NO and EOT=YES is not allowed with SRBEXIT. Do not let EOM default to NO.

If you specify SRBEXIT, you cannot also specify EXIT. See ‘Coding the Listener User Exit Routine’ in [z/OS MVS Programming: Authorized Assembler Services Guide](https://www.ibm.com/support/knowledgecenter/SSEPG1_2.4.0/com.ibm.zos.v2r4.ssp.doc/welcome.jsp) for information about SRBEXIT environment.

\_EXIT=exitrtn addr

Specifies the address of the listener user exit routine that receives control when the requested system event occurs. If you want this listener user exit routine to run in 31-bit mode, you must turn on the high order bit of the exit routine’s address; otherwise the exit gets control in 24-bit mode.

If you specify EXIT, you cannot also specify SRBEXIT. See Exit Routine Environment in [z/OS MVS Programming: Authorized Assembler Services Guide](https://www.ibm.com/support/knowledgecenter/SSEPG1_2.4.0/com.ibm.zos.v2r4.ssp.doc/welcome.jsp) for information about EXIT Environment.

\_PARM=parm addr

Specifies the address of a parameter list that the ENF listener can use to pass parameters to the listener user exit routine. This address is stored into the third word of a six-word data structure pointed to by register 1 on entry to the listener user exit routine.
The fifth word of the six-word data structure is the address of the area mapped by the IEFENFSG macro. If the signal for which your listen exit is invoked originated on another system, the area mapped IEFENFSG will identify the target system. The sixth word of the data structure is reserved for possible ALET-qualification of the address mapped by the IEFENFSG macro.

You can specify either PARM=parm.addr or PARM=parm.data.

\*PARM=parm.data

Specifies a fullword of data that is stored into the third word of a six-word data structure pointed to by register 1 on entry to the listener user exit routine. Use PARM to pass data to either a standard or an SRB listener user exit routine.

The six-word data structure pointed to by register 1 on entry to the listener user exit routine:
- Address of parameter list supplied by the system for this event code
- Fullword of zeros
- Fullword of data specified by the PARM parameter of the listen request that established the listen exit
- Fullword of zeros
- Address of a parameter list mapped by the IEFENFSG macro
- Fullword of zeros

\*EOT=YES

Specifies that, if the task that issued the listen request ends, ENF no longer passes control to the listener user exit routine when the specified event occurs. EOT=YES is valid only in TCB mode with EOM=YES.

\*EOT=NO

Specifies that, if the task that issued the listen request ends, ENF continues to pass control to the listener user exit routine when the specified event occurs. EOT=NO is the default.

\*EOM=YES

Specifies that, if the address space that issued the listen request ends, ENF no longer passes control to the listener user exit routine when the specified event occurs.

\*EOM=NO

Specifies that, if the address space that issued the listen request ends, ENF continues to pass control to the listener user exit routine when the specified event occurs. EOM=NO is valid only in TCB mode with EOT=NO. EOT=NO is the default. If you specify SRBEXIT, do not let EOM default to NO.

\*XSYS=NO
\*XSYS=YES

Specifies whether this listen exit is to receive signals originating from other systems in the sysplex. XSYS=NO, the default, specifies that the listen exit is to receive only signals originating from the local system. XSYS=YES specifies that the listen exit is to receive signals from other systems in the sysplex as well those originating locally. XSYS=YES is valid only for those event codes that are defined to ENF (on the system where the listen request is established), as capable of cross-system notification. For more information about listening for system events, see z/OS MVS Programming: Authorized Assembler Services Guide.

\*PLISTVER=2
\*PLISTVER=3
ENFREQ macro

,PLISTVER=MAX
Specifies the version of the parameter list to be generated by ENFREQ. Note that MAX may be specified instead of a number, and the parameter list will be of the largest size currently supported. This size may grow from release to release (thus possibly affecting the amount of storage needed by your program). If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as that will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form. When PLISTVER is omitted, the default is the lowest version which allows all of the parameters specified on the invocation to be processed.

The parameter list field that identifies the version number of the macro is only set when the standard or list form is used, or when PLISTVER is explicitly specified. Be sure that the resulting parameter list version number covers all the keys that you use.

The following listen request keywords require the version 3 (or higher) parameter list:
• BITQUAL
• BITCOMPARE
• FLTRBLK
• MASEXIT
• XSYS

,RELATED=(value)
An optional parameter that specifies information used to self-document macros by ‘relating’ functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and can be any values.

ENF event codes and meanings
The following characteristics vary depending on the event for which you are listening.

Event code
Identifies the event

Qualifier
Further defines the specific event for which you would like to listen

Parameter list
Passes information about the event to the listener user exit

Exit type
Specifies the type of the listener user exit routine, which can be either EXIT or SRBEXIT

Cross-system capable
Specifies whether the exit is to receive signals from other systems in the sysplex
### ENFREQ macro

#### Table 4. ENF macro event codes

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Notes:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. The input save area and the information area (registers 13 and 1) point to areas above 16M. When specifying ENFREQ REQUEST=LISTEN, make sure that the exit routine (EXIT keyword) gets control in AMODE 31.</td>
<td>The defined QUAL values are:</td>
<td>Mapped by SIV1V2V3 DSECT within macro CSRSIIDF. This area contains the current information that would be returned by the CSRSI service when all data is requested (a request type of CSRL_TYPE_V1CPC_Machine plus CSRL_TYPE_V2CPC_LPAR plus CSRL_TYPE_V3CPC_VM), with the exception of the fields whose names begin with &quot;SI00PCCA&quot;. If the SI00PCCAxxx fields are needed, the CSRSI service can be called. The SIV1V2V3 area is in 31-bit storage.</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td></td>
<td>2. This exit only gets control in task mode in ASID 1.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. Event 20 “listen” exits should avoid issuing dynamic allocation (SVC99) calls.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>The system or an operator varied a device online.</td>
<td>Corresponds to the UCBTYP field in the UCB data area. The bytes in the qualifier correspond to the bytes in UCBTYP as follows: First byte = UCBDVCLS Second byte = UCBUNTYP Third byte = UCBBYT2 Fourth byte = UCBBYT1</td>
<td>Mapped by IEEFVARY</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td></td>
<td>An operator can vary a device online by using the VARY command. For more information about the VARY command, see z/OS MVS System Commands.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>The system or an operator varied a device offline.</td>
<td>Corresponds to the UCBTYP field in the UCB data area. The bytes in the qualifier correspond to the bytes in UCBTYP as follows: First byte = UCBDVCLS Second byte = UCBUNTYP Third byte = UCBBYT2 Fourth byte = UCBBYT1</td>
<td>Mapped by IEEFVARY</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td></td>
<td>An operator can vary a device offline by using the VARY command. For more information about the VARY command, see z/OS MVS System Commands.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>The system or an operator unloaded a DASD or tape volume.</td>
<td>Corresponds to the UCBTYP field in the UCB data area. The bytes in the qualifier correspond to the bytes in UCBTYP as follows: First byte = UCBDVCLS Second byte = UCBUNTYP Third byte = UCBBYT2 Fourth byte = UCBBYT1</td>
<td>Mapped by IEZEU1LD</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td></td>
<td>An operator can unload a DASD volume by issuing the VARY command. For more information about the VARY command, see z/OS MVS System Commands.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>A dynamic device reconfiguration (DDR) swap occurred.</td>
<td>None</td>
<td>8-byte parameter list. The first four bytes contain the address of the UCB for the device that was the source of the swap event. The second four bytes contain the address of the UCB for the device that was the target of the swap event.</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td></td>
<td>A DDR swap moves or swaps a demountable volume from a failed device to another available device. For information about the SWAP command, which enables an operator to perform a DDR swap, see z/OS MVS System Commands.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>The system or an operator placed a device in pending offline status.</td>
<td>Corresponds to the UCBTYP field in the UCB data area. The bytes in the qualifier correspond to the bytes in UCBTYP as follows: First byte = UCBDVCLS Second byte = UCBUNTYP Third byte = UCBBYT2 Fourth byte = UCBBYT1</td>
<td>Mapped by IEEFVARY</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td></td>
<td>An operator can place a device in offline status by issuing the VARY command. For more information about this command, see z/OS MVS System Commands.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>The system or an operator placed a volume online so that it would be available for system use.</td>
<td>Corresponds to the UCBTYP field in the UCB data area. The bytes in the qualifier correspond to the bytes in UCBTYP as follows: First byte = UCBDVCLS Second byte = UCBUNTYP Third byte = UCBBYT2 Fourth byte = UCBBYT1</td>
<td>Mapped by IEEFVARY</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td></td>
<td>An operator can place a volume online by issuing the VARY command. For more information about this command, see z/OS MVS System Commands.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>A configuration change that involves deleting a device or deleting a path to a device was requested or was rejected.</td>
<td>None</td>
<td>Mapped by IOSDDCCD</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td>32</td>
<td>A configuration change was successful.</td>
<td>None</td>
<td>Mapped by IOSDDCCD</td>
<td>EXIT / NO</td>
</tr>
</tbody>
</table>
| 33         | One of the following changes to the hardware configuration of a device occurred:  
  • A device is added or deleted from the hardware configuration definition or a device is attached or detached with the VM ATTACH or DETACH command. The I/O subchannel corresponding to the device’s UCB is connected or disconnected.  
  • A device is made available because the channel path to the device is reestablished.  
  • The description of a device is added, deleted, or changed. The self-description information is stored in a configuration data record (CDR). A change to a CDR is always a delete followed by an add. Use timestamps to determine the correct sequence.  
  • The HyperPAV mode of operation for a logical control unit is changed.  
  • A change in state has occurred for a PCIe device.  
  • A device requires monitoring. | BYTE 1 Device class (Byte 3 from UCBTYP)  
BYTE 2 Reserved  
BYTES 3-4 Qualifier number | Mapped by IOSDDACH | EXIT / NO |

Each qualifier number designates a type of change, such as I/O subchannel change, device available, a configuration data record (CDR) change, or a HyperPAV mode change. Along with each qualifier number is a qualifier number-dependent mapping in the IOSDDACH mapping macro, which designates fields specific to the type of change.

The following ENF signal 33 subtypes are issued for PAV-alias devices:

- DACHIO
- DACHIORA
- DACHCCDR
- DACHPAV

For each of these subtypes, if the signal applies to a device in the alternate subchannel set, the issuer will fill in a new field in the DACH subtype for the subchannel set identifier and change the subtype according to the information in z/OS MVS Data Areas in the z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).

ENF 33 is issued once for the logical control unit when its HyperPAV mode is changed. The following ENF 33 subtype fields are updated to uniquely identify this event:

- DACHDEVCE='CU'
- DACHTRAN='TRAN'
- DACHTRN_MODE='TRAN'
- DACHQN=X'0008'
- DACH_TRAN_CU=control unit that is changing
- DACH_TRAN_MODE=target mode of operation

For specific field definitions, see the IOSDDACH macro in z/OS MVS Data Areas in the z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).

The ENF signal 33 subtype DACHPCIE (X'0009') is issued for the change in state of a PCIe device. For this subtype, the following value is set in the DACHTYPE field by the issuer of the signal:

DACHTYPEPCIE (PCIE).

For this subtype, the DACHQN field is set to X'0009' (PCIe device event).
## ENFREQ macro

Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
</table>
| 33 (cont.) | For this subtype, the following data is supplied in the DACHQUALD field by the issuer of the signal:  
• DACH_PCIE_PFID (4 bytes): The PFID of PCIe device involved in the event.  
• DACH_PCIE_DEVID (2 bytes): The device ID of PCIe device involved in the event.  
• DACH_PCIE_VENDID (2 bytes): The vendor ID of PCIe device involved in the event.  
• DACH_PCIE_EVENT (1 byte): The device event code:  
  1 = The device is going online.  
  2 = The device is going offline. | | | |

ENF signal 33 subtype DACHMONC (DACHTYPE = DACHMONC) is issued when a change in device monitoring is requested. This signal may be generated for secondary devices monitored for HyperSwap® configurations for which I/O operations may begin to be started. This signal may also be received when devices that were previously identified by ENF 33 subtype DACHMONC no longer require monitoring.

Programs such as RMF™ may choose to monitor this ENF 33 signal to know when to begin collecting data for these devices which might otherwise see only insignificant amounts of I/O activity. For the DACHMONC subtype:

• For a device that requires monitoring, the following fields are set:
  
  **DACH_IO_QUAL**
  
  Set to
  
  DACH_IO_QUAL_MONC_ON
  
  **DACH_IO_DEVN**
  
  Device number
  
  **DACH_IO_SSID**
  
  Subchannel set identifier
  
  **DACH_IO_DTP**
  
  Contents of the UCBTYP field from the UCB
  
  **DACHUCBC**
  
  Device class
  
  **DACHQN**
  
  Either DACHIO or DACHIO_AS

Each device receives a separate signal to begin monitoring. When monitoring is requested, the UCBCMONR bit is set on in the UCB.

• When one or more devices no longer require monitoring, a single ENF 33 DACHTYPE = DACHMONC, DACHQN = DACHIO signal is given with DACH_IO_QUAL = DACH_IO_QUAL_MONC_OFF. This is typically done after a HyperSwap occurs or when a configuration is purged from the HyperSwap manager. When this signal is received, a UCBCSCAN can be done to detect devices for which the UCBCMONR bit is no longer set on. DACH_IO_DEVN and DACH_IO_SSID are not used for this signal.
Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>One of the following XES or XCF events has occurred:</td>
<td>None</td>
<td>Mapped by IXCYENF</td>
<td>EXIT or SRBEXIT / NO</td>
</tr>
<tr>
<td></td>
<td>• New coupling facility resources have become available on this system.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Requests to connect with IXCONN that previously failed might now</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>succeed because of this new coupling facility resource.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• A specific structure has become available for use. Requests to connect to</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>the structure with IXCONN that previously failed might now succeed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>because of this new coupling facility resource.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• A system has joined the sysplex. The system name and ID are presented to</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>the user.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• A system has been partitioned from the sysplex. The system name and ID are</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>presented to the user.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• A CF definition with a SITE specified has been added or an existing CF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SITE specification has changed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Note that the listener user exit routine for event code 35 can run in SRB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mode.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>The system wrote a record to the logrec data set or the logrec log stream.</td>
<td>The specific logrec record type value is used as the qualifier for each ENF event code 36 signal. Note that no signal is issued for record types X'99'.</td>
<td>Mapped by IFBENF36</td>
<td>EXIT or SRBEXIT / NO</td>
</tr>
<tr>
<td></td>
<td>ENF passes to the listener user exit routine a parameter list containing the</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>record information.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>For details about the contents of the parameter list, see IFBENF36 in z/OS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MVS Data Areas in the z/OS Internet library:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Additional considerations for listeners of this code include the following:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The mapping does not indicate whether an IBM or non-IBM program caused the</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>record to be written to logrec.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ENF does not suppress duplicate ENF signals sent to the listener. The</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>listener must be aware of instances where a program loop causes the same</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>software record to be recorded in logrec multiple times, thus causing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENF to issue duplicate signals.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ENF does not filter software records based on any criteria including ABEND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>codes.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>One of the following SMF accounting-related events occurred:</td>
<td>ENF37Q00 SMF address space was initialized.</td>
<td>Mapped by IFAENF37</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td></td>
<td>• SMF was initialized</td>
<td>ENF37Q01 SMF address space ended.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SMF ended</td>
<td>ENF37Q02 SMF INTVAL parameter changed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SMF SYNCVAL parameter changed</td>
<td>ENF37Q03 SMF SYNCVAL parameter changed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SMF interval expired</td>
<td>ENF37Q04 SMF interval expired.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SMF interval sync processing disabled</td>
<td>ENF37Q05 SMF interval sync processing disabled.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SMF event driven interval occurred</td>
<td>ENF37Q06 SMF event driven interval occurred.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>For information about these accounting-related events, see z/OS MVS System Management Facilities (SMF).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>One of the following automatic restart manager events occurred:</td>
<td>None</td>
<td>Mapped by IXCYAREN</td>
<td>SRBEXIT / NO</td>
</tr>
<tr>
<td></td>
<td>• A job or task started or was restarted, and has registered or reregistered as an element of the automatic restart manager.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• An element notified the system that it is ready to accept work.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• An element has deregistered with the automatic restart manager.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• This system has acquired (or regained) access to the automatic restart management couple data set. Batch jobs and started tasks may now register as elements of the automatic restart manager.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• An element has been deregistered with the automatic restart manager.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>A JES2 subsystem either completed initialization or ended normally. (Note that ENF code 40 does not reflect situations in which JES2 abends.)</td>
<td>ENF40_INIT A JES completed initialization.</td>
<td>Mapped by IEFENF40</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td></td>
<td>ENF passes to the listener user exit routine a parameter list that identifies the JES2 subsystem. For details about the contents of the parameter list, see IEFENF40 in z/OS MVS Data Areas in the z/OS Internet library (<a href="http://www.ibm.com/systems/z/os/zos/ibmxr">http://www.ibm.com/systems/z/os/zos/ibmxr</a>)</td>
<td>ENF40_TERM A JES ended normally</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>A workload management (WLM) event occurred. The following qualifiers for ENF code 41 are provided.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>BYTE 1</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Policy change was initiated.</td>
<td>WLMENF11</td>
<td>Mapped by IWMRENF1</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td></td>
<td>2 Policy change completed.</td>
<td>WLMENF12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 Policy change failed.</td>
<td>WLMENF13</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>BYTE 2</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reserved.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>BYTE 3</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Workload activity reporting failed and has begun recovery.</td>
<td>WLMENF31</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 Workload activity reporting recovery was successful.</td>
<td>WLMENF32</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 Workload activity reporting recovery was not successful.</td>
<td>WLMENF33</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>BYTE 4</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 WLM service definition was successfully installed.</td>
<td>WLMENF41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>A new copy of workload management sampled address space information is available via IWMRQRY.</td>
<td></td>
<td>Four byte parameter containing the length of the storage required to hold the information. A listener can pass this length to IWMRQRY in the ANSLEN parameter and save issuing IWMRQRY to determine the length.</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td>44</td>
<td>A configuration change involving paths to a coupling facility has occurred.</td>
<td>None</td>
<td>Mapped by IXLYCFSE</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td>45</td>
<td>The SMSVSAM server address space has been initialized or reinitialized after a failure. Any subsystem that lost connection to the service provider address space can now reconnect.</td>
<td>None</td>
<td>Mapped by IDAENF45</td>
<td>SRBEXIT / NO</td>
</tr>
<tr>
<td>46</td>
<td>z/OS UNIX System Services has been initialized or reinitialized.</td>
<td>None</td>
<td>None</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td>47</td>
<td>DAE has detected that the threshold for completed or suppressed dumps, related to a particular symptom string, has been reached.</td>
<td>None</td>
<td>Mapped by ADYENF</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td>48</td>
<td>A status change has occurred within system logger. The events issued by ENF 48 are issued to all systems in the sysplex. For a description of using ENF event 48 for system logger, see z/OS MVS Programming: Authorized Assembler Services Guide. For a description of the events mapped by the IXGENF macro, see z/OS MVS Data Areas in the z/OS Internet library.</td>
<td>None</td>
<td>Mapped by IXGENF</td>
<td>SRBEXIT / YES</td>
</tr>
<tr>
<td>49</td>
<td>The logrec output recording medium has been changed by the SETLOGRC command.</td>
<td>None</td>
<td>IFBNTASM</td>
<td>SRBEXIT / NO</td>
</tr>
</tbody>
</table>
## ENFREQ macro

### Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>One of the following types of GRS information: • Resource contention information • RNL change effects on user jobs • GRS mode change information</td>
<td>The qualifier (QUAL parameter) has the following format: BYTE 1 Type of signal information: x'01' Contention data; x'02' RNL changes; x'03' Mode changes</td>
<td>Contention data: ISGE51CN</td>
<td>EXIT or SRBEXIT / YES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE 2</td>
<td>Always x'00'.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE 3</td>
<td>Varies with type of signal (value of BYTE1): x'00' Normal contention; x'01' Waitless contention</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE 4</td>
<td>Varies with type of signal (value of BYTE1): x'01' Local events; x'02' Global events; x'03' Recovery events</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The defined QUAL values are:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Qualifier Information type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>A LNKLST set has been activated. A LNKLST set can be activated at IPL through a PROGxx LNKLST statement, or through SET PROG=xx or SETPROG LNKLST operator commands. For information about PROGxx, see z/OS MVS Initialization and Tuning Reference. For commands, see z/OS MVS Commands.</td>
<td>None</td>
<td>Mapped by CSVDLENF</td>
<td>EXIT or SRBEXIT / NO</td>
</tr>
</tbody>
</table>

Note that the listener user exit routine for event code 51 can run in SRB mode.

Event code 51 can generate large numbers of events in short periods of time. The listener user exit routine for event code 51 must handle the volume of events. See z/OS MVS Programming: Authorized Assembler Services Guide for a description of system services to avoid when writing listener user exits.

Supports Filter Block (FLKBLOCK) listeners. Mapped by ISGYELF. The filter block reason codes for EnfReq RC=X'68' is in field ISGYELF_ReasonCode. The mapping also includes constants for the various values of the reason code.

Supports Filter Block (FLKBLOCK) listeners. Mapped by ISGYELF. The filter block reason codes for EnfReq RC=X'68' is in field ISGYELF_ReasonCode. The mapping also includes constants for the various values of the reason code.

Contingency data: ISGE51CN
RNL data: ISGE51RN
Mode change data: None

EXIT or SRBEXIT / YES

None

Mapped by CSVDLENF
EXIT or SRBEXIT / NO
### Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
</table>
| 53         | • A Sysplex Timer (ETR) configuration change occurred.  
            • A change to the local time offset occurred. | None | 8-byte parameter list  
            • Bytes 1 and 2 indicate a configuration change to the Sysplex Timer (ETR).  
            • Byte 3, if non-zero, indicates a change to the local time offset. Possible values are:  
              0 Sysplex Timer configuration has changed.  
              1 Local time offset has changed.  
              2 Leap second offset has changed.  
              3 Both local time offsets have changed. | EXIT / NO |
| 55         | The system resource manager (SRM) has detected a significant MVS image event, which is being signalled. The qualifiers and parameters further define the event. | | Mapped by IRAENF55 | EXIT or SRBEXIT / NO |

| ENF55QLF_REAL_SHORTAGE (X'80000000') | Too many fixed frames in storage; issued when IRA400E occurs. |
| ENF55QLF_REAL_SHORTAGE_RELIED (X'40000000') | Pageable storage shortage due to excessive fixed storage relieved; issued when IRA402I occurs. |
| ENF55QLF_REAL_WARNING (X'20000000') | Pageable storage warning that indicates there are many fixed frames in storage; issued when IRA405I occurs. |
| ENF55QLF_AUX_CRITICAL_SHORTAGE (X'08000000') | Too many slots allocated in the AUX subsystem. It is a critical shortage and is issued when IRA201E occurs. |
| ENF55QLF_AUX_SHORTAGE (X'02000000') | Too many slots allocated in the AUX subsystem. It is issued when IRA200E occurs. |
| ENF55QLF_AUX_SHORTAGE_RELIED (X'02000000') | AUX Storage shortage due to excessive slots relieved. It is issued when IRA202I occurs. |
| ENF55QLF_AUX_WARNING (X'01000000') | AUX Storage usage warning that indicates there are many slots allocated in the AUX subsystem. It is issued when IRA205I occurs. |
| ENF55QLF_SCM_HIGH_USAGE (X'00040000') | High usage of storage-class memory (SCM). Issued when IRA250I occurs. |
## ENFREQ macro

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
<tbody>
<tr>
<td>55 (cont.)</td>
<td>ENF55QLF_SCIONLIFICATION (X'00000000')</td>
<td>High usage of storage-class memory (SCM) relieved. Issued when IRA252I occurs.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55 (cont.)</td>
<td>ENF55QLF_AFQ.SHORTAGE (X'00000000')</td>
<td>Available frame queue shortage. Not enough frames on the available frame queue.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55 (cont.)</td>
<td>ENF55QLF_AFQ.SHORTAGE.RELIEVED (X'00000000')</td>
<td>Available frame queue shortage relieved.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55 (cont.)</td>
<td>ENF55QLF_PREF.SHORTAGE (X'00000000')</td>
<td>Preferred frame queue shortage. Not enough frames on the preferred frame queue.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55 (cont.)</td>
<td>ENF55QLF_PREF.SHORTAGE.RELIEVED (X'00000000')</td>
<td>Preferred frame queue shortage relieved.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>Workload management has changed an attribute of a job.</td>
<td>WLMENF56.QUAL.RESET</td>
<td>Mapped by IWMRENF2</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td>57</td>
<td>The state of a workload management scheduling environment has been altered.</td>
<td>WLMENF57.NORMAL_SCHEDENV_CHANGE</td>
<td>Mapped by IWMRENF57</td>
<td>EXIT / NO</td>
</tr>
<tr>
<td>57</td>
<td>The state of a workload management scheduling environment has been altered.</td>
<td>WLMENF57.RECOVERY_SCHEDENV_CHANGE</td>
<td>Mapped by IWMRENF57</td>
<td>EXIT / NO</td>
</tr>
</tbody>
</table>
Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_PURGE</td>
<td>The data set was purged.</td>
<td>Mapped by IAZENF58</td>
</tr>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_SELECT</td>
<td>The data set was selected.</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_DESELECT_PROCESSED</td>
<td>The data set was processed.</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_DESELECT_NOT_PROCESSED</td>
<td>The data set is no longer selected, disposition was not updated.</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_DESELECT_NOT_PROCESSED_HELD</td>
<td>The data set is no longer selected, disposition was not updated, and data set is held.</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_DESELECT_ERROR</td>
<td>An error resulting in a system level hold occurred.</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_EOL_OK</td>
<td>End of data set notification occurred — successful.</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_EOL_ERROR</td>
<td>End of data set notification occurred — unsuccessful.</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_JOB_CHANGE</td>
<td>A job status change occurred.</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_TOKEN_CHANGE</td>
<td>The client token has changed.</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_INSTANCE</td>
<td>Addition instance of data set created.</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_GRP_SELECT</td>
<td>Data set group select.</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the “Listening for Events” section of the “JES Client/Server Print Interface” chapter in z/OS JES Application Programming.</td>
<td>ENF58.Q_GRP_DESELECT</td>
<td>Data set group deselect.</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>A TRACE TT command has been accepted.</td>
<td>ENF60.VAL</td>
<td></td>
<td>Mapped by ITZENF60</td>
</tr>
<tr>
<td>61</td>
<td>The capacity of the MVS image or CEC has changed.</td>
<td>WLMENF61_CAPACITY_CHANGE</td>
<td></td>
<td>Mapped by IWMENF61</td>
</tr>
</tbody>
</table>
| 62         | A RACF SETROPTS RACLIST command has affected in-storage profiles used for authorization requests in a class designated as SIGNAL=NO or SIGNAL(NO) in the RACF class descriptor table. The class affected is in the parameter list in field IRR_ENFCLASS. The qualifier (QUAL) has the following format:
- **BYTE 1**: X’80’ SETROPTS RACLIST has taken place.
- **BYTE 2**: X’80’ SETROPTS RACLIST REFRESH has taken place.
- **BYTE 3**: X’80’ SETROPTS NORACLIST has taken place. | | Mapped by IRRPENFP in SYS1.MACLIB. | EXIT or SRBEXIT / NO |
| 63         | A permanent error was detected on a HyperSwap capable device. | None | | Mapped by IOSDE63R | EXIT / NO |
### ENFREQ macro

**Table 4. ENF macro event codes (continued)**

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>One of the following events occurred:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The capacity of a storage volume has changed.</td>
<td>The qualifier (QUAL parameter) has the following format:</td>
<td>Mapped by IECENF64</td>
<td>EXIT / YES</td>
</tr>
<tr>
<td></td>
<td>• The VTOC or INDEX of a direct access volume has been extended or moved to a new location.</td>
<td>BYTE 1 Type of signal information:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The VTOC index of a storage volume has been built. The index indicates a direct access volume has changed from an OS format VTOC (OSVTOC) to an indexed format VTOC (IXVTOC).</td>
<td>BYTE 2 Varies with event</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The content of the volume has changed due to a full volume copy or restore operation.</td>
<td>BYTE 3 Always X’00’</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DS8K recovery scenario occurs either on primary or secondary PPRC disk subsystem. DS8K signals z/OS on all paths via the Storage Controller Health Message attention status.</td>
<td>BYTE 4 Varies with event</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• One or more devices in the logical subsystem has a PPRC state change.</td>
<td>The defined QUAL values are:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Qualifier Information type</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’01’ Volume events</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’02’ DASD volume capacity changed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’03’ VTOC updated (moved or extended)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’04’ VTOC index is built</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’05’ Full volume copy or restore</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’06’ LSS event</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’07’ Storage controller health (LSS) event</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X’08’ Summary (LSS) event - PPRC state change</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>System REXX event has occurred.</td>
<td>X'80000000'</td>
<td>The AXR address space has initialized. AXREXX can be invoked.</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'40000000'</td>
<td>The AXR address space has terminated. Subsequent AXREXX invocations will be rejected.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'20000000'</td>
<td>The AXR address space has reached its threshold of ACTIVE+WAITING AXREXX requests. No more requests will be accepted until the number of requests drops to an acceptable level.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'10000000'</td>
<td>The number of ACTIVE+WAITING AXREXX requests has dropped to an acceptable level. AXREXX requests are now being accepted.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'08000000'</td>
<td>The number of ACTIVE+WAITING AXREXX requests is high and is nearing the level where subsequent requests will be rejected.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'04000000'</td>
<td>The number of extents in the REXXLIB concatenation exceeds the system limit. See <a href="https://www.ibm.com">z/OS DFSMS User Data Sets</a> for details. If this condition is detected during System REXX initialization, System REXX terminates; otherwise, no new AXREXX requests will be accepted.</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>One of the following IBM Health Checker for z/OS events has occurred:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBM Health Checker for z/OS has become available.</td>
<td>X'80000000'</td>
<td>The defined BITQUAL values are:</td>
<td>Mapped by HZSZENF</td>
</tr>
<tr>
<td></td>
<td>IBM Health Checker for z/OS has terminated and is not available.</td>
<td>X'40000000'</td>
<td>IBM Health Checker for z/OS is available. Field Enf067_BitQual_Available in the HZSZENF mapping macro.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM Health Checker for z/OS has terminated and is not available. Field Enf067_BitQual_NotAvailable in the HZSZENF mapping macro.</td>
<td></td>
</tr>
</tbody>
</table>
### ENFREQ macro

**Table 4. ENF macro event codes (continued)**

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
</table>
| 68         | One of the following BCPii events has occurred:  
- A change in BCPii status has occurred.  
- A hardware communication error has occurred.  
- A hardware event has occurred. | The defined QUAL values are:  
Qualifier Information type  
X'01000001' BCPii is available.  
X'01000002' BCPii is not available.  
X'0201yy' A hardware communication error has occurred and CPC events might have been lost. *yy* denotes the type of error:  
01 A temporary error, some events might have been lost.  
02 A permanent error, no more events are delivered.  
03 Communication to the CPC has been established or re-established. Event delivery from this CPC will now commence or re-commence.  
X'03xx0yy' A hardware event has occurred. *xx* denotes the event source:  
01 CPC  
02 Image  
*yy* denotes the event. | Mapped by HWICIASM and HWICIC | EXIT / NO |

The defined BITQUAL values are:  
Qualifier Information type  
X'01nnnnnn' N/A  
X'0201nnnn' Bytes 1-17 CPC name, padded with hexadecimal zeros  
X'0301nnnn' Bytes 1-17, CPC name, padded with hexadecimal zeros  
X'0302nnnn' Bytes 18-24, image name, padded with hexadecimal zeros  
X'0302nnnn' Bytes 1-17, CPC name, padded with hexadecimal zeros  
X'0302nnnn' Bytes 18-24, image name, padded with hexadecimal zeros
### Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
</tr>
</thead>
</table>
| 68 | One of the following BCPii events has occurred:  
  - A change in BCPii status.  
  - A hardware communication error has occurred.  
  - A hardware event has occurred. | Hardware Event Codes:  
  - Code: 'W' x A command response has been received.  
  - Code: 'W2' x An object status change has occurred.  
  - Code: 'W3' x An object name change has occurred.  
  - Code: 'W4' x The activation profile of the object has changed.  
  - Code: 'W5' x A new object was created.  
  - Code: 'W6' x An object was deleted.  
  - Code: 'W7' x An object entered or left an exception state.  
  - Code: 'W8' x A Console application has started.  
  - Code: 'W9' x A Console application has ended.  
  - Code: 'WA' x An operating system message has been received.  
  - Code: 'WB' x A hardware message has been received.  
  - Code: 'WC' x A hardware message has been deleted.  
  - Code: 'WD' x A capacity change event has been received.  
  - Code: 'WE' x A capacity record change has occurred.  
  - Code: 'WF' x A security event has been logged.  
  - Code: 'W10' x An image has entered a disabled wait state.  
  - Code: 'W11' x A power change event has been received. | Mapped by HWICIASM and HWICIC |
| 70 | The state of a job (batch, STC or TSU) owned by JES has changed. The job may have been selected for processing, completed processing, changed phase (including changes to execution phase job class), or been purged. | ENF70_SELECT  
Job was selected.  
ENF70_DESELECT  
Job was processed.  
ENF70_CHANGE  
Job queued to new phase of processing.  
ENF70_PURGE  
Job was purged. | Mapped by IAZENF70 |
| 71 | A RACF command has affected a user’s group connections which may affect his or her resource authorization.  
The user affected is in the parameter list in field IRR_ENF2USER.  
The group affected is in the parameter list in field IRR_ENF2GROUP.  
Control flags that are used to provide greater granularity for the listeners are in the parameter list in field IRR_ENF2Flags. | The qualifier (QUAL) has the following format:  
BYTE 1  
X'80' CONNECT command  
X'40' REMOVE command  
X'20' ALTUSER REVOKE command  
X'10' DELUSER command  
X'08' DELGROUP command  
BYTES 2 - 4  
Reserved | Mapped by IRRPENF2 (See z/OS Security Server RACF Data Areas) |

ENFREQ macro
### ENFREQ macro

#### Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
</table>
| 72         | Volume status information for SMS. The listener user exit routine for event code 72 can run in SRB mode. Event code 72 can generate large numbers of events in short periods of time. The listener user exit routine for event code 72 must be able to handle the volume of events. | The qualifier (QUAL) has the following format: 
**BYTE 1** Type of signal information: X'01' Volume status
**BYTE 2** Always X'00'
**BYTES 3 - 4** Varies with event. The defined QUAL values are: 
**Qualifier Information type** X'01000001'
ENF72_OVER_THRESHOLD_AM_Y. Volume over threshold when storage group is defined using AM=Y (Auto Migrate, yes). | Volume Status information: IGDENV72VL Mapped by IGDENV72 | EXIT or SRBEXIT / YES |
| 73         | The SETLOAD xx, IEASYM command has completed successfully. The local system's symbol table has been updated. | None | None | EXIT / NO |
| 78         | The state of a job (batch, STC or TSU) owned by JES2 has changed. The job has completed processing. | ENF78_JOB_NOTIFY | Mapped by IZAENF78 | EXIT / YES |
### ENFREQ macro

#### Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>A RACF command has modified a profile such that a user's authorization to the resources it protects may be affected.</td>
<td>The qualifier (IRR_ENF3_QualCode) has the following format:</td>
<td>EXIT or SRBEXIT / YES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The user affected is in the parameter list in field IRR_ENF3_UserID.</td>
<td>BYTE 1</td>
<td>EXIT or SRBEXIT / YES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The class in which the modified profile belongs is in the parameter list in field IRR_ENF3_ClassName.</td>
<td></td>
<td>EXIT or SRBEXIT / YES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The length of the affected profile name is in the parameter list in field IRR_ENF3_ProfName_Length.</td>
<td></td>
<td>EXIT or SRBEXIT / YES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The name of the affected profile is in the parameter list in field IRR_ENF3_ProfName.</td>
<td></td>
<td>EXIT or SRBEXIT / YES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Control flags that are used to provide greater granularity for the listeners are in the parameter list in field IRR_ENF3_Flags.</td>
<td></td>
<td>EXIT or SRBEXIT / YES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For the PERMIT RACF command processor, there may be additional information regarding:</td>
<td></td>
<td>EXIT or SRBEXIT / YES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The type of Conditional Access, a numerical value that is in the parameter list in field IRR_ENF3_PERMIT_WHEN_Cond.</td>
<td>BYTE 2 - 4</td>
<td>EXIT or SRBEXIT / YES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The Conditional Access List Entry. The length of the Conditional Access Name and the Conditional Access Name itself is in the parameter list in the fields: IRR_ENF3_CACLNName_Length IRR_ENF3_CACLNName</td>
<td></td>
<td>EXIT or SRBEXIT / YES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For the RDEFINE and RALTER RACF command processors, there may be additional information in the ADDMEM and DELMEM lists. The number of elements in the list, the length of the list, and the offset to the list are in the parameter list in the fields: IRR_ENF3_ADDMEML_Member# IRR_ENF3_DELMEML_Member# IRR_ENF3_ADDMEML_Length IRR_ENF3_DELMEML_Length IRR_ENF3_ADDMEML_Offset IRR_ENF3_DELMEML_Offset</td>
<td></td>
<td>EXIT or SRBEXIT / YES</td>
<td></td>
</tr>
</tbody>
</table>

The qualifier (IRR_ENF3_QualCode) has the following format:

**BYTE 1**

- X'80' PERMIT command
- X'40' RDEFINE command
- X'20' RALTER command
- X'10' DELETE command

**BYTES 2 - 4**

Reserved

Mapped by IRRPENF3 (See ENFREQ Exit)
### ENFREQ macro

#### Table 4. ENF macro event codes (continued)

<table>
<thead>
<tr>
<th>Event code</th>
<th>Description</th>
<th>Qualifier</th>
<th>Parameter list passed to the user exit</th>
<th>Exit type / Cross-system capable</th>
</tr>
</thead>
</table>
| 80         | One of the following z/OS Communication Server events has occurred:  
- The rpcbind server has initialized.  
- The rpcbind server is stopping.  
Notes:  
1. ENF80_RPC DSECT maps the RPCBIND event.  
2. Use the ENF80_RPC_FLAGS to determine if the rpcbind server is initializing or ending.  
3. When flag ENF80_RPCINIT is on, RPC applications can register with RPCBIND.  
4. When flag ENF80_RPCTERM is on, the rpcbind server is stopping.  
5. ENF80_RPC DSECT includes the jobname of the rpcbind server that generated the event. | ENF80_RPC_EVENT | Mapped by EZAENF80 | SRBEXIT / NO |

#### Return codes

When ENFREQ macro returns control to your program, GPR 15 contains a return code.

#### Table 5. Return Codes for the ENFREQ Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | Meaning: ENFREQ processing completed successfully.  
Action: None |
| 04                      | Meaning: Program error. An identical LISTEN request already exists. A request is considered a duplicate if its QUAL, QMASK, EXIT, BITQUAL, and BITCOMPARE parameter values are the same as those specified for an existing request.  
Action: None. The request is already established. |
| 0C                      | Meaning: Program error (invalid parameter list) The ENFREQ failed for one of the following reasons:  
- The length of the parameter list is incorrect.  
- The specified ACTION code is not valid.  
- The specified EVENT code is not valid.  
- The caller specified ACTION=LISTEN, and the EXIT address is zero.  
- The caller specified ACTION=DELETE, and the DTOKEN field is zero.  
Action: After checking and correcting the program environment parameters, retry the request. If the parameters are correct, check to see if you inadvertently overlaid the control parameter list. |
| 10                      | Meaning: System error. This return code is for IBM diagnostic purposes only.  
Action: Record the return code, and supply it to the appropriate IBM support personnel. |
### Table 5. Return Codes for the ENFREQ Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
</table>
| 14                      | **Meaning**: Environmental error. Your program issued the ENFREQ macro before the system initialized ENF.  
                          | **Action**: Retry the request. If the problem persists, record the return code and supply it to the appropriate IBM support personnel. |                                                                                           |
| 18                      | **Meaning**: Environmental error. The system cannot obtain storage for your request.  
                          | **Action**: Rerun your program one or more times. If the problem persists, check with the operator to see if another user in the installation is causing the problem, or if the entire installation is experiencing storage constraint problems. |                                                                                           |
| 1C                      | **Meaning**: Program error. The DTOKEN parameter does not represent any LISTEN request that is currently active. ENF does not perform a DELETE.  
                          | **Action**: Verify that the DTOKEN on the DELETE request matches the DTOKEN from the LISTEN request. Retry the DELETE request with the correct DTOKEN. |                                                                                           |
| 20                      | **Meaning**: Program error. An abend occurred in the Listen Exit code.  
                          | **Action**: If a dump was produced for the abend, examine it and correct the programming error. |                                                                                           |
| 3C                      | **Meaning**: Program error. EOT=YES was specified on an ENFREQ listen request while the issuer of the ENFREQ request was running in SRB mode.  
                          | **Action**: Either specify EOT=NO or delete the EOT keyword from the ENFREQ macro invocation. |                                                                                           |
| 46                      | **Meaning**: Program error. The SRBEXIT keyword was specified on an ENFREQ listen request for an event code that does not allow SRBEXIT.  
                          | **Action**: Verify that the listen request is for the correct event code. If so, replace the SRBEXIT keyword with the EXIT keyword and ensure that the listen exit resides in common storage. |                                                                                           |
| 48                      | **Meaning**: Program error. The EXIT keyword was specified on an ENFREQ listen request for an event code that does not allow EXIT.  
                          | **Action**: Verify that the listen request is for the correct event code. If so, replace the EXIT keyword with the SRBEXIT keyword. |                                                                                           |
| 4A                      | **Meaning**: Program error. The keyword combination of EOT=YES and EOM=NO was specified on an ENFREQ listen request. This combination is incorrect.  
                          | **Action**: Change the EOM specification to YES or the EOT specification to NO. |                                                                                           |
| 4C                      | **Meaning**: Program error. EOM=NO and SRBEXIT were specified on an ENFREQ listen request. This combination is incorrect.  
                          | **Action**: Change the EOM specification to YES or do not use SRBEXIT. |                                                                                           |
| 4E                      | **Meaning**: Program error. An ENF request specified XSYS=YES for an event code that does not support sysplex-wide notification.  
                          | **Action**: Verify that the ENF request is for the correct event code. If so, specify XSYS=NO (or allow the XSYS parameter to default to XSYS=NO). |                                                                                           |
| 50                      | **Meaning**: System error. Sysplex-wide notification is not available, because of a system initialization problem. ENF listeners will receive notifications originating from only the system where the listen exit was established.  
                          | **Action**: Report the problem to the operator and the system programmer. The cross-system signalling capability will remain unavailable until the next system IPL. |                                                                                           |
Table 5. Return Codes for the ENFREQ Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 52                      | **Meaning:** Program error. Sysplex-wide notification services were requested for an action type other than listen.  
                           **Action:** Verify that your program is not overwriting the parameter list, and that the execute form of the macro correctly addresses the parameter list. |
| 54                      | **Meaning:** Program error. An ENF request specified invalid comparison instructions for the bit-mapped qualifier.  
                           **Action:** Verify that your program is not overwriting the parameter list, and that the execute form of the macro correctly addresses the parameter list. |
| 60                      | **Meaning:** Program error. An ENF request specified FLTRBLK for an event code that does not support listener filter blocks.  
                           **Action:** Verify that the ENF request is for the correct event code. If so, do not specify FLTRBLK. |
| 64                      | **Meaning:** Program error. An ENF request specified FLTRBLK. It was specified for an event code that does support listener filter blocks, but the block was not accessible by the owner of that particular event code.  
                           **Action:** Ensure that the event-specific listener filter block occupies accessible storage of sufficient length. |
| 68                      | **Meaning:** Program error. An ENF request specified FLTRBLK. It was specified for an event code that does support listener filter blocks, and the block was accessible by the owner of that particular event code, but the filter parameters are incorrect.  
                           **Action:** Check the parameters specified in the FLTRBLK. If the event-specific mapping includes a reason code, use its value to assist with the problem determination. |

**Example 1**

Set up and load into common storage the SMFLST00 listener user exit routine, which gains control only if the qualifier equals ENF37Q00.

Note that the qualifiers are declared in the IFAENF37 mapping macro. The ENFREQ macro specifies QMASK=ALL which requests that all four bytes of the qualifier mask are used in the qualifier comparison.

* Load ENF Listen Exit (SMFLST00) into common storage and save address.  
  * SMFL00@ contains the address of the listener user exit routine that resides in common storage  
  * ST R00,SMFL00@  

* Issue LISTEN Request for SMF Event Code (Qualifier ENF37Q00)  
  * ENFREQ ACTION=LISTEN, -- Function  
    CODE=ENFC37, -- Event Code  
    EXIT=(R02), -- Exit Address  
    QUAL=ENF37Q00, -- Qualifier Value  
    QMASK=ALL, -- Qualifier Mask (Full Word)  
    ESTBNME=THISMOD, -- Establisher Name  
    EXITNME=SMFLST00, -- Exit Name  
    DTOKEN=SMFL00T -- Returned Token Field  
  * Check the return code from ENFREQ - if not zero issue message  
  *  
  * Local variables
  DATAAREA DSECT

62  z/OS V2R1.0 MVS Authorized Assembler Services Reference EDT-DXG
ENFREQ macro

SMFL000 DS A
SMFL001 DS F
ENFPTR DS A
*
* - Local constants
SMFLSTEN CSECT
DS 0F
ENFC37 EQU 37
THISMOD DC CL8'SMFLSTEN'
SMFLST00 DC CL8'SMFLST00'
*
* - External control blocks
CVT DSECT=YES
SMFLSTEN CSECT
IFAEENF37
DATAAREA DSECT
IEFENFPM
LENODATA EQU *+DATAAREA
*

Note that the IFAENF37 macro includes the following declarations:

SYSECT CSECT Control Section for Constants
ENF37Q00 DC X'80000000' SMF Active
ENF37Q01 DC X'40000000' SMF Terminated

Example 2

Set up and load into storage the ENFLST01 listener user exit routine. This listener user exit routine receives a parameter from the ENF listener when the specified event occurs. The listener user exit runs in the address space of the listener and is deleted when the address space that issued the listen request ends.

* Load ENF Listen Exit (ENFLST01) into storage and save address.
* ENFL01@ contains the address of the listener user exit routine.
*
* Issue LISTEN Request for Event Code 35
L R02,ENFL01@
ENFREQ ACTION=LISTEN, -- Function +
CODE=ENFC35, -- Event Code +
SRBEXIT=(R02), -- Exit Address +
PARM=LPARM, -- Parameter +
EOT=YES, -- End-of-task delete indicator +
EOM=YES, -- End-of-memory delete indicator +
ESTBMME=THISMOD, -- Establisher Name +
EXITNAME=ENFLST01, -- Exit Name +
DTOKEN=ENFL01T -- Returned Token Field
*
* Check the return code from ENFREQ - if not zero issue message
*
* - Local variables
DATAAREA DSECT
ENFL010 DS A
ENFL011 DS F
ENFPTR DS A
LPARM DS CL16
*
* - Local constants
ENFLSTEN CSECT
DS 0F
ENFC35 EQU 35
THISMOD DC CL8'ENFLSTEN'
ENFLST01 DC CL8'ENFLST01'
*
* - External control blocks
CVT DSECT=YES
**DELETE option**

**Syntax**

The standard form of the ENFREQ macro for ACTION=DELETE is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ENFREQ.</td>
</tr>
<tr>
<td>ENFREQ</td>
<td></td>
</tr>
<tr>
<td>ACTION=DELETE</td>
<td></td>
</tr>
<tr>
<td>,CODE=event code</td>
<td>event code: Decimal digit.</td>
</tr>
<tr>
<td>,DTOKEN=dtoken</td>
<td>dtoken: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,RELATED=(value)</td>
<td>value: Any text.</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

**ACTION=DELETE**

A required parameter that specifies that you want to delete an existing request to listen for a specified event. When a program issues ENFREQ with the ACTION=DELETE parameter, ENF either deletes the listen request immediately if the listener user exit has completed, or waits until the listener user exits completes. Because the listener user exit might not have completed processing at the time the delete request is issued, do not release the listener user exit's storage.

**,CODE=event code**

A required parameter that specifies the ENF event for which a program no longer needs notification. The event code can be any of the decimal codes listed in Table 4 on page 44.
The required parameter that identifies the specific listen request you are deleting. The system returned the token when you issued the ACTION=LISTEN request.

An optional parameter that specifies information used to self-document macros by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and can be any valid coding values.

Return and reason codes

For the return codes, in hexadecimal, from the ENFREQ macro see "Return codes" on page 60.

On systems running z/OS V2R1 or higher, for return code of 0 or 28 (X'1C') from a ACTION=DELETE request, a reason code is provided in access register 15:

0 The ACTION=DELETE request has completed. The listen exit is not executing and will not be called again.
1 The ACTION=DELETE request is pending. The listen exit may be executing or may be called again.

The reason code provides a way to determine when it is safe to free or reuse storage containing the exit or used by the exit. Storage can be safely freed or reused when the first ACTION=DELETE request provides return code 0 and reason code 0, or after the first ACTION=DELETE request provides return code 0 and reason code 1, a subsequent ACTION=DELETE request provides return code 28 (X'1C') and reason code 0.

Because there is no way to determine when it is safe to free or reuse storage containing the exit or used by the exit on systems running z/OS V1R13 or earlier, such storage should never be freed or reused.

ENFREQ ACTION=LISTEN - List form

Use the list form of the ENFREQ macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro constructs a parameter list that the execute form of the macro can use or modify.

Syntax

The list form of the ENFREQ macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ENFREQ.</td>
</tr>
<tr>
<td>ENFREQ</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ENFREQ.</td>
</tr>
</tbody>
</table>
## ENFREQ macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION=LISTEN</td>
<td></td>
</tr>
<tr>
<td>,CODE=event code</td>
<td>event code: Decimal digit.</td>
</tr>
<tr>
<td>,MASEXIT=No</td>
<td>Default: MASEXIT=NO</td>
</tr>
<tr>
<td>,MASEXIT=YES</td>
<td></td>
</tr>
<tr>
<td>,FLTRBLK=filter block addr</td>
<td>filter block addr: the address of the filter block</td>
</tr>
<tr>
<td>,QUAL=qualifier</td>
<td>qualifier: A constant value</td>
</tr>
<tr>
<td>,QMASK=qmask keywords</td>
<td>qmask keywords: BYTE1, BYTE2, BYTE3, BYTE4, ALL, NONE. Default: QMASK=NONE</td>
</tr>
<tr>
<td>,BITQUAL=bitqual</td>
<td>bitqual: name of a 32-byte field, hexadecimal numeric value (X'xxx'),</td>
</tr>
<tr>
<td>,BITCOMPARE=SUBSET</td>
<td>Default: BITCOMPARE=SUBSET</td>
</tr>
<tr>
<td>,BITCOMPARE=INTERSECT</td>
<td></td>
</tr>
<tr>
<td>,BITCOMPARE=EQUAL</td>
<td></td>
</tr>
<tr>
<td>,SRBEXIT=exitrtn addr</td>
<td>exitrtn addr: A-type address.</td>
</tr>
<tr>
<td>,EXIT=exitrtn addr</td>
<td>exitrtn addr: A-type address.</td>
</tr>
<tr>
<td>,PARM=parm addr</td>
<td>parm addr: A-type address.</td>
</tr>
<tr>
<td>,PARM=parm data</td>
<td>parm data: a fullword of data</td>
</tr>
<tr>
<td>,EOT=NO</td>
<td>Default: EOT=NO.</td>
</tr>
<tr>
<td>,EOT=YES</td>
<td></td>
</tr>
<tr>
<td>,EOM=NO</td>
<td>Default: EOM=NO.</td>
</tr>
<tr>
<td>,EOM=YES</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=2</td>
<td>Default: Version implied by keywords</td>
</tr>
<tr>
<td>,PLISTVER=3</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,RELATED=(value)</td>
<td>value: Is any text.</td>
</tr>
<tr>
<td>,XSYS=NO</td>
<td>Default: XSYS=NO.</td>
</tr>
<tr>
<td>,XSYS=YES</td>
<td></td>
</tr>
</tbody>
</table>
Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,MF=L</td>
<td>Specifies the list form of the ENFREQ macro with ACTION=LISTEN.</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained under the standard form of the ENFREQ macro with ACTION=LISTEN, with the following exceptions:

\,MF=L

Specifies the list form of the ENFREQ macro with ACTION=LISTEN.

**ENFREQ ACTION=LISTEN - Execute form**

Use the execute form of the ENFREQ macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro uses or modifies the parameter list that the list form built.

**Syntax**

The execute form of the ENFREQ macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ENFREQ.</td>
</tr>
<tr>
<td>ENFREQ</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ENFREQ.</td>
</tr>
<tr>
<td>ACTION=LISTEN</td>
<td></td>
</tr>
<tr>
<td>,CODE=,event code</td>
<td>event code: Decimal digit.</td>
</tr>
<tr>
<td>,DTOKEN=,dtoken addr</td>
<td>dtoken addr: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ESTBNME=,estab name</td>
<td>estab name: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,EXITNME=,exitrtn name</td>
<td>exitrtn name: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,FLTRBLK=,filter block addr</td>
<td>filter block addr: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,QUAL=,qualifier</td>
<td>qualifier: A four-byte value.</td>
</tr>
</tbody>
</table>
ENFREQ macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.QMASK=qmask keywords</code></td>
<td>qmask keywords: BYTE1, BYTE2, BYTE3, BYTE4, ALL, NONE. Default: QMASK=NONE</td>
</tr>
<tr>
<td><code>.BITQUAL=bitqual</code></td>
<td>bitqual: name of a 32-byte field, hexadecimal numeric value (X'xxx'), or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.BITCOMPARE=SUBSET</code></td>
<td>Default: BITCOMPARE=SUBSET</td>
</tr>
<tr>
<td><code>.BITCOMPARE=INTERSECT</code></td>
<td></td>
</tr>
<tr>
<td><code>.BITCOMPARE=EQUAL</code></td>
<td></td>
</tr>
<tr>
<td><code>.SRBEXIT=exitrtn addr</code></td>
<td>exitrtn addr: A-type address, or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.EXIT=exitrtn addr</code></td>
<td>exitrtn addr: A-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.PARM=parm addr</code></td>
<td>parm addr: A-type address, or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.PARM=parm data</code></td>
<td>parm data: a fullword of data</td>
</tr>
<tr>
<td><code>.EOT=NO</code></td>
<td>Default: EOT=NO.</td>
</tr>
<tr>
<td><code>.EOT=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>.EOM=NO</code></td>
<td>Default: EOM=YES.</td>
</tr>
<tr>
<td><code>.EOM=NO</code></td>
<td></td>
</tr>
<tr>
<td><code>.PLISTVER=2</code></td>
<td>Default: Version implied by keywords</td>
</tr>
<tr>
<td><code>.PLISTVER=3</code></td>
<td></td>
</tr>
<tr>
<td><code>.PLISTVER=MAX</code></td>
<td></td>
</tr>
<tr>
<td><code>.RELATED=(value)</code></td>
<td>value: Is any text.</td>
</tr>
<tr>
<td><code>.XSYS=NO</code></td>
<td>Default: XSYS=NO.</td>
</tr>
<tr>
<td><code>.XSYS=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(E,list addr)</code></td>
<td>list addr: RX-type address or address in register (2) - (12).</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained under the standard form of the ENFREQ macro with ACTION=LISTEN, with the following exceptions:

`.MF=(E,list addr)`

Specifies the execute form of the ENFREQ macro with ACTION=LISTEN.

list addr specifies the area that the system uses to store the parameters.
### ENFREQ ACTION=DELETE - List form

Use the list form of the ENFREQ macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro constructs a parameter list that the execute form of the macro can use or modify.

#### Syntax

The list form of the ENFREQ macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ENFREQ.</td>
</tr>
<tr>
<td>ENFREQ</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ENFREQ.</td>
</tr>
</tbody>
</table>

#### Parameters

The parameters are explained under the standard form of the ENFREQ macro with ACTION=DELETE, with the following exceptions:

- `MF=L`

  Specifies the list form of the ENFREQ macro with ACTION=DELETE.

### ENFREQ ACTION=DELETE - Execute form

Use the execute form of the ENFREQ macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro uses or modifies the parameter list that the list form built.

#### Syntax

The execute form of the ENFREQ macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name: Symbol. Begin name in column 1.</td>
</tr>
</tbody>
</table>
ENFREQ macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>One or more blanks must precede ENFREQ.</td>
</tr>
<tr>
<td>ENFREQ</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ENFREQ.</td>
</tr>
<tr>
<td>ACTION=DELETE</td>
<td></td>
</tr>
<tr>
<td>,CODE=event code</td>
<td>event code: Decimal digit.</td>
</tr>
<tr>
<td>,DTOKEN=dtoken addr</td>
<td>dtoken addr: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,RELATED=(value)</td>
<td>value: Any text.</td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td>list addr: RX-type address or address in register (2) - (12).</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained under the standard form of the ENFREQ macro with ACTION=DELETE, with the following exceptions:

,MF=(E,list addr)

Specifies the execute form of the ENFREQ macro with ACTION=DELETE.

list addr specifies the area that the system uses to contain the parameters.
Chapter 4. ENQ — Request control of a serially reusable resource

Description

ENQ assigns control of one or more serially reusable resources to a task. If any of the resources are not available, the task might be placed in a wait condition until all of the requested resources are available. Once control of a resource has been assigned to a task, it remains with that task until one of the programs running under that task issues a DEQ macro to release the resource or the task terminates.

You can request either shared or exclusive use of a resource. ENQ identifies the resource by a pair of names, the qname and the rname, and a scope value. The scope value determines what other tasks, address spaces, or systems can use the resource. All programs that share the resource must use the qname, rname, and scope value consistently.

Use ENQ with RET=TEST to determine the status of the resource. Return codes tell whether the resource is immediately available or in use, and whether control has been previously requested by the active task in another ENQ macro.

ENQ with the MASID and MTCB parameters allows a further conditional control of a resource. One task, called the “issuing task” can issue an ENQ macro for a resource specifying the ASID and TCB of another task, called the “matching task”. MTCB and MASID parameters are specified with RET=HAVE, RET=TEST, or ECB to provide additional return codes. If the issuing task does not receive control of the resource, it may receive a return code indicating that the resource is controlled by the matching task. Upon receiving this return code, the issuing task could use the resource, if serialization between itself and the matching task has been prearranged through a protocol.

Global resource serialization counts and limits the number of concurrent resource requests from an address space. If an unconditional ENQ (an ENQ that uses the RET=NONE option) causes the count of concurrent resource requests to exceed the limit, the caller ends abnormally with a system code of 'X’538’. For more information, see the topic on limiting concurrent requests for resources in z/OS MVS Programming: Assembler Services Guide.

Unless you specify otherwise, when a global resource serialization complex is initialized, global resource serialization searches the SYSTEM inclusion resource name list (RNL) and the SYSTEMS exclusion RNL for every resource specified with a scope of SYSTEM or SYSTEMS. A resource whose name appears on one of these RNLs might have its scope changed from the scope that appears on the macro. To prevent RNL processing, use the RNL=NO parameter. See z/OS MVS Planning: Global Resource Serialization for additional information about RNL processing.

The ENQ macro is also described in z/OS MVS Programming: Assembler Services Reference ABE-HSP with the exception of the SMC, ECB, TCB, MASID, and MTCB parameters. For information on using the ENQ macro to serialize resources, see the z/OS MVS Programming: Authorized Assembler Services Guide.
**Environment**

The requirements for callers of ENQ are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
</table>
| Minimum authorization:        | Problem state with any PSW key. For the SMC, ECB, TCB, MASID, and MTCB parameters or when the specified qname is ADRDFRAG, ADRDSN, ARCTG, BWODSN, SYSZ*, SYSCTLG, SYSDNS, SYSIEA01, SYSIEECT, SYSIEFSD, SYSGG, SYSGG2, SYSSW, SYSSAM, or SYSVTOC, the authorization must be **one of the following**:
|                               | • Supervisor state
|                               | • PSW key 0-7
|                               | • APF-authorized.                                                                                                                                                                                                 |
| Dispatchable unit mode:       | Task                                                                                                                                                                                                 |
| Cross memory mode:            | For LINKAGE=SVC: PASN=HASN=SASN                                                                                                                                                                                                 |
|                              | For LINKAGE=SYSTEM: Any PASN, Any HASN, Any SASN                                                                                                                                                                                              |
| AMODE:                        | 24- or 31- or 64-bit                                                                                                                                                                                                                         |
| ASC mode:                     | Primary                                                                                                                                                                                                                                    |
| Interrupt status:             | Enabled for I/O and external interrupts                                                                                                                                                                                                   |
| Locks:                        | No locks held                                                                                                                                                                                                                              |
| Control parameters:          | Control parameters must be in the primary address space. Except for the TCB, all parameters can reside above 16 megabytes.                                                                                                                   |

**Programming requirements**

None.

**Restrictions**

See "Avoiding Interlock" in [z/OS MVS Programming: Assembler Services Guide](https://www.ibm.com/docs/en/zos) to ensure that you are following the protocols required to prevent the interlock.

Issuing two ENQ macros for the same resource without an intervening DEQ macro causes the task to end abnormally, unless the second ENQ designates RET=TEST, USE, CHNG, or HAVE. If the task ends, either normally or abnormally, while the task still has control of any serially reusable resources, all requests made by this task automatically have DEQ processing performed for them. If resource input addresses are incorrect, the task abnormally ends.

The caller cannot have an EUT FRR established.

There are some considerations to be aware of when using enclaves for tasks that serialize resources using the ENQ macro. For details, see "Using ENQ/DEQ or Latch Manager Services With Enclaves" in [z/OS MVS Programming: Workload Management Services](https://www.ibm.com/docs/en/zos).

**Input register information**

Before issuing the ENQ macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.
Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• If you specify RET=TEST, RET=USE, RET=CHNG, RET=HAVE, or ECB:</td>
</tr>
<tr>
<td></td>
<td>If all return codes for the resources named in the ENQ macro are 0,</td>
</tr>
<tr>
<td></td>
<td>register 15 contains 0. If any of the return codes are not 0, register 15</td>
</tr>
<tr>
<td></td>
<td>contains the address of a storage area containing the return codes.</td>
</tr>
<tr>
<td></td>
<td>• Otherwise: Used as a work register by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the ENQ macro is described as follows.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ENQ.</td>
</tr>
<tr>
<td>ENQ</td>
<td>ENQ</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ENQ.</td>
</tr>
<tr>
<td>(</td>
<td></td>
</tr>
<tr>
<td>qname addr</td>
<td>qname addr: A-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,</td>
<td></td>
</tr>
<tr>
<td>,rname addr</td>
<td>rname addr: A-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
**ENQ macro**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,</td>
<td>Default: E</td>
</tr>
<tr>
<td>,E</td>
<td></td>
</tr>
<tr>
<td>,S</td>
<td></td>
</tr>
<tr>
<td>,</td>
<td></td>
</tr>
<tr>
<td>,rname length</td>
<td>rname length: symbol, decimal digit, or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td>Default: assembled length of rname</td>
</tr>
<tr>
<td></td>
<td>Note: Code rname length if rname addr is a register.</td>
</tr>
<tr>
<td>,</td>
<td></td>
</tr>
<tr>
<td>,STEP</td>
<td>Default: STEP</td>
</tr>
<tr>
<td>,SYSTEM</td>
<td></td>
</tr>
<tr>
<td>,SYSTEMS</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td></td>
</tr>
<tr>
<td>,RET=CHNG</td>
<td>Default: RET=NULL</td>
</tr>
<tr>
<td>,RET=HAVE</td>
<td></td>
</tr>
<tr>
<td>,RET=TEST</td>
<td></td>
</tr>
<tr>
<td>,RET=USE</td>
<td></td>
</tr>
<tr>
<td>,RET=NULL</td>
<td></td>
</tr>
<tr>
<td>,SMC=NULL</td>
<td>Default: SMC=NULL</td>
</tr>
<tr>
<td>,SMC=STEP</td>
<td></td>
</tr>
<tr>
<td>,ECB=ecb addr</td>
<td>ecb addr: A-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,TCB=tcb addr</td>
<td>tcb addr: A-type address or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td>Note: Do not specify ECB with RET. You can specify ECB and TCB together. If TCB is specified without ECB, you must specify RET=CHNG, TEST or USE.</td>
</tr>
<tr>
<td>,MASID=matching-asid addr</td>
<td>matching-asid addr: A-type address or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td>Note: MTCB is required with MASID. Do not specify SMC or TCB with MASID.</td>
</tr>
<tr>
<td>,MTCB=matching-tcb addr</td>
<td>matching-tcb addr: A-type address or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td>Note: MASID is required with MTCB.</td>
</tr>
<tr>
<td>,RNL=YES</td>
<td>Default: RNL=YES</td>
</tr>
</tbody>
</table>
Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,RNL=NO</td>
<td></td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>value: any valid macro keyword specification.</td>
</tr>
<tr>
<td>,LINKAGE=SVC</td>
<td>DEFAULT: LINKAGE=SVC</td>
</tr>
<tr>
<td>,LINKAGE=SYSTEM</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

- **qname addr**
  Specifies the address of an 8-character name. The name can contain any valid hexadecimal character. Every program issuing a request for a serially reusable resource must use the same qname, rname, and scope to represent the resource. Some names, such as those beginning with certain letter combinations (SYSZ for example), are used to protect system resources by requiring that the issuing program be in supervisor state, or system key, or APF-authorized. Authorized programs should use a restricted qname (as described under Minimum authorization in the Environment topic of this chapter) to prevent interference from unauthorized programs.

  **Note:** See z/OS MVS Diagnosis: Reference for a list of major and minor ENQ/DEQ names and the resources that issue the ENQ/DEQ.

- **rname addr**
  Specifies the address of the name used together with qname to represent a single resource. The name must be from 1 to 255 bytes long, can be qualified, and can contain any valid hexadecimal character.

- **E, S**
  Specifies whether the request is for exclusive (E) or shared (S) control of the resource. If the resource is modified while under control of the task, the request must be for exclusive control; if the resource is not modified, the request should be for shared control.

- **rname length**
  Specifies the length of the rname. If this parameter is omitted, the system uses the assembled length of the rname. To override the assembled length, specify this parameter.

  The value you can code depends on whether or not you also specify MASID and MTCB:
  - If you specify MASID and MTCB, you can code a value between 1 and 128.
  - If you do not specify MASID and MTCB, you can code a value between 1 and 255.

  In either case, you can specify 0, which means that the length of the rname must be contained in the first byte at the rname addr.
ENQ macro

\texttt{,STEP}
\texttt{,SYSTEM}
\texttt{,SYSTEMS}

Specifies the scope of the resource.

STEP specifies that the resource can be used only within an address space. If STEP is specified, a request for the same \texttt{qname} and \texttt{rname} from a program in another address space denotes a different resource.

SYSTEM specifies that the resource can be used by programs in more than one address space.

SYSTEMS specifies that the resource can be shared between systems.

STEP, SYSTEM, and SYSTEMS are mutually exclusive and do not refer to the same resource. If two macros specify the same \texttt{qname} and \texttt{rname}, but one specifies STEP and the other specifies SYSTEM or SYSTEMS, they are treated as requests for different resources.

\texttt{)}

Specifies the end of the resource description.

Notes on specifying multiple resources on one ENQ request:

- Within a single set of parentheses, you can repeat the \texttt{qname addr}, \texttt{rname addr}, type of control, \texttt{rname length}, and the scope until there is a maximum of 255 characters, including the parentheses.

- The following parameters apply to all the resources you specify on the request: RET, SMC, ECB, TCB, MASID, MTCB, and RNL.

\texttt{,SMC=NONE}
\texttt{,SMC=STEP}
\texttt{,ECB=ecb addr}

\texttt{,RET=CHNG}
\texttt{,RET=HAVE}
\texttt{,RET=TEST}
\texttt{,RET=USE}
\texttt{,RET=NONE}

Specifies the type of request for the resources named on the ENQ request.

\texttt{CHNG}

The status of the resource specified is changed from shared to exclusive control. When RET=CHNG is specified, the exclusive|shared (E|S) parameter is overridden. This parameter ensures that the request will be exclusive regardless of the other parameter.

\texttt{HAVE}

Control of the resources is requested conditionally; that is, control is requested only if a request has not been made previously for the same task.

\texttt{TEST}

The availability of the resources is to be tested, but control of the resources is not requested.

\texttt{USE}

control of the resources is to be assigned to the active task only if the resources are immediately available. If any of the resources are not available, the active task is not placed in a wait condition.

\texttt{NONE}

Control of all the resources is unconditionally requested.

See "Return and reason codes" on page 79 for an explanation of the return codes for these requests.

\texttt{,SMC=NONE}
\texttt{,SMC=STEP}
\texttt{,ECB=ecb addr}
 Specifies optional parameters available to the system programmer:

SMC specifies that the set must-complete function is not to be used (NONE) or that it is to set as non-dispatchable other tasks for the step until the requesting task has completed its operations on the resource (STEP).


Do not use SMC or RET with ECB.

When SMC=STEP is specified with RET=HAVE and the requesting task already has control of the resource, the SMC function is turned on and the task continues to control the resource.

SMC and TCB are mutually exclusive with the MASID parameter; therefore, hexadecimal return codes 20, 24, 28, and 44 will not be given by an ENQ using the SMC or TCB operands.

The return codes and status of the set must-complete function for the various RET specifications are as follows:

<table>
<thead>
<tr>
<th>RET Parameter</th>
<th>Hexadecimal Code</th>
<th>SMC Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>RET=CHNG</td>
<td>0</td>
<td>on</td>
</tr>
<tr>
<td>RET=CHNG</td>
<td>4</td>
<td>off</td>
</tr>
<tr>
<td>RET=CHNG</td>
<td>8</td>
<td>off</td>
</tr>
<tr>
<td>RET=CHNG</td>
<td>14</td>
<td>off</td>
</tr>
<tr>
<td>RET=HAVE</td>
<td>0</td>
<td>on</td>
</tr>
<tr>
<td>RET=HAVE</td>
<td>8</td>
<td>on</td>
</tr>
<tr>
<td>RET=HAVE</td>
<td>14</td>
<td>off</td>
</tr>
<tr>
<td>RET=HAVE</td>
<td>18</td>
<td>off</td>
</tr>
<tr>
<td>RET=TEST</td>
<td>0</td>
<td>off</td>
</tr>
<tr>
<td>RET=TEST</td>
<td>4</td>
<td>off</td>
</tr>
<tr>
<td>RET=TEST</td>
<td>8</td>
<td>off</td>
</tr>
<tr>
<td>RET=TEST</td>
<td>14</td>
<td>off</td>
</tr>
<tr>
<td>RET=USE</td>
<td>0</td>
<td>on</td>
</tr>
<tr>
<td>RET=USE</td>
<td>4</td>
<td>off</td>
</tr>
<tr>
<td>RET=USE</td>
<td>8</td>
<td>off</td>
</tr>
<tr>
<td>RET=USE</td>
<td>14</td>
<td>off</td>
</tr>
<tr>
<td>RET=USE</td>
<td>18</td>
<td>off</td>
</tr>
</tbody>
</table>

ECB specifies the address of an ECB, and conditionally requests all of the resources named in the macro. If the return code for one or more requested resources is hexadecimal 4 or 24 and the request is not nullified by a corresponding DEQ, the ECB is posted when all the requested resources (specifically, those that initially received a return code of 4 or 24) are assigned to the requesting task.

If the ECB parameter is an A-type address, the address is the name of the fullword that is used as an ECB. If the operand is a register, then the register contains the address of the ECB.
ENQ macro

Note: The ECB must reside in storage that is addressible from the caller's home address space.

TCB specifies a register that points to a TCB or specifies the address of a fullword on a fullword boundary that points to a TCB on whose behalf the ENQ is to be done. If TCB is specified, one of the following must also be specified:
  - RET=TEST
  - RET=USE
  - RET=CHNG
  - ECB

Note: The TCB resides in storage below 16 megabytes in the caller's address space.

\masid=matching-asid addr

Specifies the matching task (by defining a matching ASID) for the ENQ, if it is used together with the MTCB parameter. MASID defines the ASID of a task that may be using a resource desired by the caller. If the MASID parameter is an A-type address, the address is the name of a fullword containing the ASID. If the operand is a register, then the register contains the ASID.

\m tcb=matching-tcb addr

Specifies the matching task (by defining a matching TCB) for the ENQ, if used together with the MASID parameter. MTCB defines the TCB of a task that may be using a resource desired by the caller. If the MTCB parameter is an A-type address, the address is the name of a fullword containing the TCB. If the operand is a register, then the register contains the TCB.

If the task specified by the MASID and MTCB parameters is not using the resource, global resource serialization gives control to the caller and returns a return code indicating whether the resource can be used. If the task specified by MASID and MTCB parameters is using the resource, global resource serialization records a request for the resource, suspends the issuing task until the resource is available, or optionally returns a return code indicating that an ECB will be posted when the resource can be used.

The MASID and MTCB parameters are specified with RET=HAVE, RET=TEST, or ECB parameters to elicit additional return codes that provide information about the owner of the resource.

See the description of the \r name length for information about specifying \r name length with MASID and MTCB.

\rnl=yes
\rnl=no

Controls global resource serialization RNL processing, which can cause the scope value of a resource to change. IBM recommends that you use the default, RNL=YES, to allow global resource serialization to perform RNL processing. Use RNL=NO when you are sure that you want the request to be processed only by global resource serialization using only the specified scope. When RNL=NO is specified the ENQ request will be ignored by alternative serialization products. Refer to \z os MVS Planning: Global Resource Serialization RNL Processing, for more information about the use of RNL=NO.

\related=value

Specifies information used to self-document macros by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.
ENQ macro

,LINKAGE=SVC
,LINKAGE=SYSTEM

Specifies the type of linkage the caller is using to invoke the ENQ service.

For LINKAGE=SVC, the linkage is through an SVC instruction. This linkage is valid only when the caller is in primary mode and the primary, home, and secondary address spaces are the same.

For LINKAGE=SYSTEM, the linkage uses a non-SVC entry. This linkage is valid in cross memory mode or in non-cross memory mode. LINKAGE=SYSTEM is intended to be used by programs in cross memory mode.

- If ECB= is specified, the ECB (not the address of the ECB) must be addressable from the home address space.
- If TCB= is specified, then the specified TCB in the home address space is associated with the resource; otherwise, the TCB in the home address space making the request is associated with the resource.

The default is LINKAGE=SVC.

ABEND codes

For only unconditional requests, the caller might encounter abend code X'138' or X'538'. For unconditional or conditional requests, the caller might encounter one of the following abend codes:

- X'238'
- X'338'
- X'438'
- X'738'
- X'838'
- X'938'

See z/OS MVS System Codes for explanations and responses for these codes.

Return and reason codes

The system provides a return code only if you specify RET=TEST, RET=USE, RET=CHNG, RET=HAVE, or ECB; otherwise, return of the task to the active condition indicates that control of the resource has been assigned or was previously assigned to the task. If all return codes for the resources named in the ENQ macro are 0, register 15 contains 0. For nonzero return codes, register 15 contains the address of a storage area containing the return codes, as shown in Figure 4 on page 80.
ENQ macro

The return codes are placed in the parameter list resulting from the macro expansion in the same sequence as the resource names in the ENQ macro.

The return codes for the ENQ macro with the RET=TEST parameter are described in Table 6.

Table 6. Return Codes for the ENQ Macro with the RET=TEST Parameter

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning:** The resource is immediately available.  
                          **Action:** None required. However, you might take some action based on your application. |
| 4                       | **Meaning:** The resource is not immediately available.  
                          **Action:** None required. However, you might take some action based on your application. |
| 8                       | **Meaning:** A previous request for control of the same resource has been made for the same task. The task has control of the resource.  
                          **Action:** None required. However, you might take some action based on your application.  
                          To determine whether the task has exclusive control or shared control of the resource, check bit 3 of flag byte 1 in the parameter list that identifies the owned resource. If bit 3 is off, the task has exclusive control; if bit 3 is on, the task has shared control. |
### Table 6. Return Codes for the ENQ Macro with the RET=TEST Parameter (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 14                      | **Meaning**: A previous request for control of the same resource has been made for the same task. The task does not have control of the resource.  
**Action**: None required. However, you might take some action based on your application. |
| 20                      | **Meaning**: The matching task (the task specified in the MASID and MTCB parameters) owns the resource.  
**Action**: None required. However, you might take some action based on your application. |

The return codes for the ENQ macro with the RET=USE parameter are described in [Table 7](#).

### Table 7. Return Codes for the ENQ Macro with the RET=USE Parameter

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning**: The active task now has control of the resource.  
**Action**: None. |
| 4                       | **Meaning**: The resource is not immediately available.  
**Action**: None required. However, you might take some action based on your application. |
| 8                       | **Meaning**: A previous request for control of the same resource has been made for the same task. The task has control of the resource.  
**Action**: None required. However, you might take some action based on your application.  
To determine whether the task has exclusive control or shared control of the resource, check bit 3 of flag byte 1 in the parameter list that identifies the owned resource. If bit 3 is off, the task has exclusive control; if bit 3 is on, the task has shared control. |
| 14                      | **Meaning**: A previous request for control of the same resource has been made for the same task. The task does not have control of the resource.  
**Action**: None required. However, you might take some action based on your application. |
| 18                      | **Meaning**: Environmental error. The limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource.  
**Action**: Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded. |

The return codes for the ENQ macro with the RET=CHNG parameter are described in [Table 8](#).

### Table 8. Return Codes for the ENQ Macro with the RET=CHNG Parameter

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning**: The status of the resource has been changed to exclusive.  
**Action**: None. |
### Table 8. Return Codes for the ENQ Macro with the RET=CHNG Parameter (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 4                       | **Meaning:** The status of the resource cannot be changed to exclusive. Other tasks share the resource.  
**Action:** None required. However, you might take some action based on your application. |
| 8                       | **Meaning:** The status of the resource cannot be changed to exclusive. Either no tasks have issued an ENQ request for the resource, or the task acquired the resource through the MASID parameter.  
**Action:** None required. However, you might take some action based on your application. |
| 14                      | **Meaning:** The status of the resource cannot be changed to exclusive. A previous request for control of the same resource has been made for the same task. The task does not have control of the resource.  
**Action:** None required. However, you might take some action based on your application. |

The return codes for the ENQ macro with the RET=HAVE parameter are described in [Table 9](#).

### Table 9. Return Codes for the ENQ Macro with the RET=HAVE Parameter

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning:** The active task now has control of the resource.  
**Action:** None. |
| 8                       | **Meaning:** A previous request for control of the same resource has been made for the same task. The task has control of the resource.  
**Action:** None required. However, you might take some action based on your application.  
To determine whether the task has exclusive control or shared control of the resource, check bit 3 of flag byte 1 in the parameter list that identifies the owned resource. If bit 3 is off, the task has exclusive control; if bit 3 is on, the task has shared control. |
| 14                      | **Meaning:** A previous request for control of the same resource has been made for the same task but that request has not yet been satisfied (such as an ENQ with RET=NONE which waits for the resource). The task does not have control of the resource.  
**Action:** None required. However, you might take some action based on your application. |
| 18                      | **Meaning:** Environmental error. The limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource.  
**Action:** Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded. |
Table 9. Return Codes for the ENQ Macro with the RET=HAVE Parameter (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 20                      | Meaning: The matching task (the task specified in the MASID and MTCB parameters) owns the resource.  
Action: The caller can use the resource, but it must ensure that the owning task does not terminate while the caller is using the resource. If the caller requested exclusive control, then this return code indicates that the matching task is the only task that currently owns the resource. If the caller requested shared control and the owning task requested shared control, this return code might indicate that a previous task had requested exclusive control. The caller must issue a DEQ macro to cancel this ENQ request. |
| 28                      | Meaning: The caller cannot obtain exclusive control of the resource using the ENQ macro with the MASID and MTCB parameters. The matching task's involvement with other tasks precludes control by the caller.  
Action: This task must not issue a DEQ macro to cancel the ENQ request. |
| 44                      | Meaning: The caller is violating a restriction of using the ENQ macro with the MASID and MTCB parameters in one or more of the following ways:  
- Another task has already issued the ENQ macro for this resource specifying the same values for the MASID and MTCB parameters  
- The MASID and MTCB parameters specify a task that acquired control of the resource by using the ENQ macro with the MASID and MTCB parameters  
- The matching task requested ownership of the resource but has not yet been granted ownership.  
Action: Do not use the resource; the caller does not have control of it. |

The return codes for the ENQ macro with the ECB parameter are described in [Table 10](#).

Table 10. Return Codes for the ENQ Macro with the ECB Parameter

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | Meaning: The active task now has control of the resource.  
Action: Do not wait on the ECB; it will not be posted. |
| 4                       | Meaning: The active task does not have control of the resource yet. The ECB will be posted when the system assigns control to that task.  
Action: Wait on the ECB if your program cannot continue processing without control of the resource. |
| 8                       | Meaning: A previous request for control of the same resource has been made for the same task. The task has control of the resource.  
Action: Do not wait on the ECB; it will not be posted.  
To determine whether the task has exclusive control or shared control of the resource, check bit 3 of flag byte 1 in the parameter list that identifies the owned resource. If bit 3 is off, the task has exclusive control; if bit 3 is on, the task has shared control. |
| 14                      | Meaning: A previous request for control of the same resource has been made for the same task. The task does not have control of the resource.  
Action: Do not wait on the ECB; it will not be posted. |
Table 10. Return Codes for the ENQ Macro with the ECB Parameter (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 18                      | **Meaning**: Environmental error. The limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource.  
**Action**: Do not wait on the ECB; it will not be posted. Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded. |
| 20                      | **Meaning**: The matching task (the task specified in the MASID and MTCB parameters) owns the resource.  
**Action**: Do not wait on the ECB; it will not be posted. The caller can use the resource, but it must ensure that the owning task does not terminate while the caller is using the resource. If the caller requested exclusive control, then this return code indicates that the matching task is the only task that currently owns the resource. If the caller requested shared control and the owning task requested shared control, this return code might indicate that a previous task had requested exclusive control. The caller must issue a DEQ macro to cancel this ENQ request. |
| 24                      | **Meaning**: The caller that specifies the ENQ macro with the MASID and MTCB parameters will have exclusive control after the ECB is posted.  
**Action**: Wait on the ECB. Once the ECB is posted, the caller may use the resource, but must ensure that the matching task does not terminate while the caller is using the resource. The caller must issue a DEQ macro to cancel the ENQ request. |
| 28                      | **Meaning**: The caller cannot obtain exclusive control of the resource using the ENQ macro with the MASID and MTCB parameters. The matching task's involvement with other tasks precludes control by the caller.  
**Action**: Do not wait on the ECB; it will not be posted. The caller must not issue a DEQ macro to cancel the ENQ request. |
| 44                      | **Meaning**: The caller is violating a restriction of using the ENQ macro with the MASID and MTCB parameters in one or more of the following ways:  
- Another task has already issued the ENQ macro for this resource specifying the same values for the MASID and MTCB parameters  
- The MASID and MTCB parameters specify a task that acquired control of the resource by using the ENQ macro with the MASID and MTCB parameters  
- The matching task requested ownership of the resource but has not yet been granted ownership.  
**Action**: Do not wait on the ECB; it will not be posted. Do not use the resource; the caller does not have control of it. |

**Example 1**

Unconditionally request exclusive control of a serially reusable resource that is known only within the address space (STEP), and set to non-dispatchable other tasks for the step until the requesting task has completed its operations on the resource.

\[\text{ENQ (MAJOR1,MINOR1,E,8,STEP),SMC=STEP}\]

**Example 2**

Conditionally request control of a resource that can be shared on behalf of another task. The resource is known by more than one address space, and is only wanted if immediately available.

\[\text{ENQ (MAJOR2,MINOR2,S,4,SYSTEM),TCB=(R2),RET=USE}\]
ENQ - List form

Use the list form of ENQ to construct a control program parameter list. You can specify any number of resources on ENQ, therefore, the number of qname, rname, and scope combinations in the list form of the ENQ macro must be equal to the maximum number of qname, rname, and scope combinations in any execute form of the macro that refers to that list form.

Syntax

The list form of the ENQ macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede ENQ.</td>
</tr>
<tr>
<td>ENQ</td>
<td></td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must follow ENQ.</td>
</tr>
<tr>
<td><code>(</code></td>
<td></td>
</tr>
<tr>
<td><code>qname addr</code></td>
<td><code>qname addr</code>: A-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,</code></td>
<td></td>
</tr>
<tr>
<td><code>rname addr</code></td>
<td><code>rname addr</code>: A-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,</code></td>
<td></td>
</tr>
<tr>
<td><code>,</code></td>
<td>Default: E</td>
</tr>
<tr>
<td><code>,E</code></td>
<td></td>
</tr>
<tr>
<td><code>,S</code></td>
<td></td>
</tr>
<tr>
<td><code>,</code></td>
<td></td>
</tr>
<tr>
<td><code>,rname length</code></td>
<td><code>rname length</code>: symbol or decimal digit.</td>
</tr>
<tr>
<td><code>,</code></td>
<td>Default: assembled length of <code>rname</code></td>
</tr>
<tr>
<td><code>,</code></td>
<td>Default: STEP</td>
</tr>
<tr>
<td><code>,STEP</code></td>
<td></td>
</tr>
<tr>
<td><code>,SYSTEM</code></td>
<td></td>
</tr>
<tr>
<td><code>,SYSTEMS</code></td>
<td></td>
</tr>
<tr>
<td><code>)</code></td>
<td></td>
</tr>
</tbody>
</table>
ENQ macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,RET=CHNG</td>
<td><strong>Default:</strong> RET=NONE</td>
</tr>
<tr>
<td>,RET=HAVE</td>
<td></td>
</tr>
<tr>
<td>,RET=TEST</td>
<td></td>
</tr>
<tr>
<td>,RET=TEST</td>
<td></td>
</tr>
<tr>
<td>,RET=NONE</td>
<td></td>
</tr>
<tr>
<td>,SMC=NONE</td>
<td><strong>Default:</strong> SMC=NONE</td>
</tr>
<tr>
<td>,SMC=STEP</td>
<td></td>
</tr>
<tr>
<td>,ECB=ecb addr</td>
<td><em>ecb addr: A-type address.</em></td>
</tr>
<tr>
<td>,TCB=0</td>
<td><strong>Note:</strong> ECB cannot be specified with RET.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> TCB or ECB must be specified on the list form if it is used on the execute form. ECB and TCB can be specified together. If you specify TCB without ECB, specify RET=CHNG, TEST or USE.</td>
</tr>
<tr>
<td>,MASID=0</td>
<td><strong>Note:</strong> MTCB is required with MASID. Do not specify SMC or TCB with MASID.</td>
</tr>
<tr>
<td>,MTCB=0</td>
<td><strong>Note:</strong> MASID is required with MTCB.</td>
</tr>
<tr>
<td>,RNL=YES</td>
<td><strong>Default:</strong> RNL=YES</td>
</tr>
<tr>
<td>,RNL=NO</td>
<td></td>
</tr>
<tr>
<td>,RELATED=value</td>
<td><em>value: any valid macro keyword specification.</em></td>
</tr>
<tr>
<td>,MF=L</td>
<td></td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained under the standard form of the ENQ macro, with the following exception:

,MF=L

- Specifies the list form of the ENQ macro.

The list form of this macro generates a prefix followed by the parameter list, however the label specified in MF=L does not include an offset prefix area. If MASID, MTCB, TCB, or ECB is specified, these labels are offset; allowance must be made for the parameter list prefix.

ENQ - Execute form
A remote control program parameter list is used in and can be modified by the execute form of the ENQ macro. The parameter list must be generated by the list form of ENQ.
## Syntax

The execute form of the ENQ macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede ENQ.</td>
</tr>
<tr>
<td>ENQ</td>
<td>One or more blanks must follow ENQ.</td>
</tr>
<tr>
<td>(</td>
<td><strong>Note</strong>: ( and ) are the beginning and end of a parameter list. The entire list is optional. If nothing in the list is desired then ( ), and all parameters between ( and ) should not be specified. If something in the list is desired, the ( ), and all parameters in the list should be specified as indicated at the left.</td>
</tr>
<tr>
<td><code>qname addr</code></td>
<td><code>qname addr</code>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,</code></td>
<td></td>
</tr>
<tr>
<td><code>,rname addr</code></td>
<td><code>rname addr</code>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,E</code></td>
<td></td>
</tr>
<tr>
<td><code>,S</code></td>
<td></td>
</tr>
<tr>
<td><code>,rname length</code></td>
<td><code>rname length</code>: symbol, decimal digit, or register (2) - (12).</td>
</tr>
<tr>
<td><code>,STEP</code></td>
<td></td>
</tr>
<tr>
<td><code>,SYSTEM</code></td>
<td></td>
</tr>
<tr>
<td><code>,SYSTEMS</code></td>
<td></td>
</tr>
<tr>
<td>)</td>
<td><strong>Note</strong>: See note opposite ( above.</td>
</tr>
<tr>
<td><code>,RET=CHNG</code></td>
<td></td>
</tr>
<tr>
<td><code>,RET=HAVE</code></td>
<td></td>
</tr>
<tr>
<td><code>,RET=TEST</code></td>
<td></td>
</tr>
<tr>
<td><code>,RET=USE</code></td>
<td></td>
</tr>
<tr>
<td><code>,RET=NONE</code></td>
<td></td>
</tr>
<tr>
<td><code>,SMC=NONE</code></td>
<td><code>ecb addr</code>: RX-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
## ENQ macro

### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.SMC=STEP</td>
<td>tcb addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>.ECB=ecb addr</td>
<td>Note: ECB cannot be specified with RET above.</td>
</tr>
<tr>
<td>.TCB=tcb addr</td>
<td>Note: ECB and TCB can be specified together. If you specify TCB without ECB, then specify RET=CHNG, TEST, or USE.</td>
</tr>
<tr>
<td>.MASID=matching-asid addr</td>
<td>matching-asid addr: RX-type address or register (2)-(12). Note: MTCB is required with MASID. Do not specify SMC or TCB with MASID.</td>
</tr>
<tr>
<td>.MTCB=matching-tcb addr</td>
<td>matching-tcb addr: RX-type address or register (2)-(12). Note: MASID is required with MTCB.</td>
</tr>
<tr>
<td>.RNL=YES</td>
<td></td>
</tr>
<tr>
<td>.RNL=NO</td>
<td></td>
</tr>
<tr>
<td>.RELATED=value</td>
<td>value: any valid macro keyword specification.</td>
</tr>
<tr>
<td>.LINKAGE=SVC</td>
<td>DEFAULT: LINKAGE=SVC</td>
</tr>
<tr>
<td>.LINKAGE=SYSTEM</td>
<td></td>
</tr>
<tr>
<td>.MF=(E, list addr)</td>
<td>list addr: RX-type address or register (1) - (12).</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the ENQ macro, with the following exceptions:

- **.MF=(E, list addr)**
  - Specifies the execute form of the ENQ macro.
  - `list addr` specifies the area that the system uses to contain the parameters.

**Note:** If ECB (or TCB) is specified in the execute form, ECB (or TCB=0) must be specified in the list form. If MASID and MTCB are specified, MASID=0 and MTCB=0 must be specified in the list form.

The list form of this macro generates a prefix followed by the parameter list, however the label specified in MF=L does not include an offset prefix area. If MASID, MTCB, TCB, or ECB is specified, these labels are offset; allowance must be made for the parameter list prefix.
Chapter 5. ESPIE — Extended SPIE

Description

The ESPIE macro extends the function of the SPIE (specify program interruption exits) macro to callers in 31-bit and 64-bit addressing mode. For additional information concerning the relationship between the SPIE and the ESPIE macros, see the information on program interruptions in "z/OS MVS Programming: Assembler Services Guide" and "z/OS MVS Programming: Authorized Assembler Services Guide".

The ESPIE macro performs the following functions using the options specified:

- Establishes an ESPIE environment (that is, identifies the interruption types that are to cause entry to the ESPIE exit routine) by executing the SET option of the ESPIE macro.
- Deletes an ESPIE environment (that is, cancels the current SPIE/ESPIE environment) by executing the RESET option of the ESPIE macro.
- Determines the current SPIE/ESPIE environment by executing the TEST option of the ESPIE macro.

The following description of the ESPIE macro also appears in "z/OS MVS Programming: Assembler Services Reference ABE-HSP" with the exception of interruption type 17. This interruption type designates page faults, and its use is restricted to programs that are APF-authorized or run in PSW key 0 - 7.

For information about programs in 64-bit addressing mode (AMODE 64), see "z/OS MVS Programming: Extended Addressability Guide".

The information documented under the following headings is provided separately for each of the three options (SET, RESET, and TEST):

- "Input Register Information"
- "Output Register Information"
- "Syntax"
- "Parameters"
- "Return and Reason Codes"
- "Examples"

The information documented in the following topics applies to all three options of the ESPIE macro (SET, RESET, and TEST):

- "Environment"
- "Programming Requirements"
- "Restrictions"
- "Performance Implications"
- "ABEND Codes"
ESPIE macro

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>To issue ESPIE without encountering an abnormal end, callers must be in problem state, with a PSW key value that is equal to the TCB assigned key, except when ESPIE RESET is issued or ESPIE SET is issued with no interruption codes specified (where key 0 supervisor state is allowed). To specify page fault processing, the caller must be APF-authorized.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24- or 31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

None.

Performance implications

Programs that need to intercept only specific hardware program check interruptions (such as arithmetic exceptions or data conversion exceptions) will find ESPIE to be more efficient than establishing an ESTAE environment to screen all abends for specific OCx abends.

ABEND codes

ESPIE might return with abend code X'46D'. See z/OS MVS System Codes for an explanation and programmer responses.

ESPIE SET option

Input register information

Before issuing the SET option of the ESPIE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain the following information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Token representing the previously active SPIE/ESPIE environment</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>
ESPIE macro

14    Used as a work register by the system
15    Return code of 0

When control returns to the caller, the access registers (ARs) contain:

Register  
Contents
0-1    Used as work registers by the system
2-13   Unchanged
14-15  Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Syntax

The standard form of the ESPIE macro with the SET option is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ESPIE.</td>
</tr>
<tr>
<td>ESPIE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ESPIE.</td>
</tr>
<tr>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>,exit addr</td>
<td>exit addr: A-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
| ,(interruptions) | interruptions: Decimal numbers 1 - 15 or 17 expressed as:  
  • single values: (2, 3, 4, 7, 8, 9, 10)  
  • ranges of values: ((2, 4), (7, 10))  
  • combinations: (2, 3, 4, (7, 10)) |
| ,PARAM=list addr | list addr: A-type address or register (2) - (12). |

Parameters

The parameters are explained as follows:

SET

Indicates that an ESPIE environment is to be established.
**ESPIE macro**

```
,exit addr
   Specifies the address of the exit routine to be given control when program
   interruptions of the type specified by interruptions occur. The exit routine will
   receive control in the same addressing mode as the issuer of the ESPIE macro.

,(interruptions)
   Indicates the interruption types that are being trapped. The interruption types
   are:

<table>
<thead>
<tr>
<th>Number</th>
<th>Interruption Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Operation</td>
</tr>
<tr>
<td>2</td>
<td>Privileged operation</td>
</tr>
<tr>
<td>3</td>
<td>Execute</td>
</tr>
<tr>
<td>4</td>
<td>Protection</td>
</tr>
<tr>
<td>5</td>
<td>Addressing</td>
</tr>
<tr>
<td>6</td>
<td>Specification</td>
</tr>
<tr>
<td>7</td>
<td>Data</td>
</tr>
<tr>
<td>8</td>
<td>Fixed-point overflow (maskable)</td>
</tr>
<tr>
<td>9</td>
<td>Fixed-point divide</td>
</tr>
<tr>
<td>10</td>
<td>Decimal overflow (maskable)</td>
</tr>
<tr>
<td>11</td>
<td>Decimal divide</td>
</tr>
<tr>
<td>12</td>
<td>Exponent overflow</td>
</tr>
<tr>
<td>13</td>
<td>Exponent underflow (maskable)</td>
</tr>
<tr>
<td>14</td>
<td>Significance (maskable)</td>
</tr>
<tr>
<td>15</td>
<td>Floating-point divide</td>
</tr>
<tr>
<td>17</td>
<td>Page fault</td>
</tr>
</tbody>
</table>
```

These interruption types can be designated as one or more single numbers, as
one or more pairs of numbers (designating ranges of values), or as any
combination of the two forms. For example, (4,8) indicates interruption types 4
and 8; ((4,8)) indicates interruption types 4 through 8.

If a program interruption type is maskable, the corresponding program mask
bit in the PSW is set to 1. If a maskable interruption is not specified, the
corresponding bit in the PSW is set to 0. Interruption types not specified above
(except for type 17) are handled by the system. The system forces an abnormal
end with the program check as the completion code. If an ESTAE-type
recovery routine is also active, the SDWA indicates a system-forced abnormal
end. The registers at the time of the error are those of the system.

**Note:** For ESPIE and SPIE - If you are using vector instructions and an
interruption of 8, 12, 13, 14, or 15 occurs, your recovery routine can check the
exception extension code (the first byte of the two-byte interruption code in the
ESPIE or PIE) to determine whether the exception was a vector or scalar type
of exception.

```
,PARAM=list addr
   Specifies the fullword address of a parameter list that is to be passed by the
caller to the exit routine.
```
Return and reason codes

None.

Example 1

Give control to an exit routine for interruption types 1 and 4. EXIT is the location of the exit routine to be given control and PARMLIST is the location of the user-parameter list to be used by the exit routine.

ESPIE SET,EXIT,(1,4),PARAM=PARMLIST

Example 2

Give control to the exit routine located at EXIT when a page fault occurs.

ESPIE SET,EXIT,(17)

ESPIE SET - List form

Use the list form of the ESPIE macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters. The list form of ESPIE is valid only for ESPIE SET.

Syntax

The list form of the ESPIE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ESPIE.</td>
</tr>
<tr>
<td>ESPIE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ESPIE.</td>
</tr>
<tr>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>,exit addr</td>
<td>exit addr: A-type address.</td>
</tr>
<tr>
<td></td>
<td>Note: This parameter must be specified on either the list or the execute form of the macro.</td>
</tr>
<tr>
<td>,(interruptions)</td>
<td>interruptions: Decimal number 1 - 15 or 17 expressed as:</td>
</tr>
<tr>
<td></td>
<td>• single values: (2, 3, 4, 7, 8, 9, 10)</td>
</tr>
<tr>
<td></td>
<td>• ranges of values: ((2, 4), (7, 10))</td>
</tr>
<tr>
<td></td>
<td>• combinations: (2, 3, 4, (7, 10))</td>
</tr>
<tr>
<td>,PARAM=list addr</td>
<td>list addr: A-type address.</td>
</tr>
<tr>
<td>,MF=L</td>
<td></td>
</tr>
</tbody>
</table>
ESPIE macro

Parameters
The parameters are explained under the standard form of ESPIE SET with the following exception:

\[,\text{MF}=L\]

Specifies the list form of the ESPIE macro.

Example
Build a nonexecutable problem program parameter list that will cause control to be transferred to the exit routine, EXIT, for the interruption types specified in the execute form of the macro. Provide the address of the user parameter list, PARMLIST.

```
LIST1 ESPIE SET,EXIT,,PARAM=PARMLIST,MF=L
```

ESPIE SET - Execute form
Use the execute form of the ESPIE macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form. The execute form of ESPIE is valid only for ESPIE SET.

Syntax
The execute form of the ESPIE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ESPIE.</td>
</tr>
<tr>
<td>ESPIE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ESPIE.</td>
</tr>
<tr>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>(exit,\text{addr})</td>
<td>exit addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>Note:</td>
<td>This parameter must be specified on either the list or the execute form of the macro.</td>
</tr>
<tr>
<td>(\text{interruptions})</td>
<td>interruptions: Decimal number 1 - 15 or 17 expressed as:</td>
</tr>
<tr>
<td></td>
<td>• single values: (2, 3, 4, 7, 8, 9, 10)</td>
</tr>
<tr>
<td></td>
<td>• ranges of values: ((2, 4), (7, 10))</td>
</tr>
<tr>
<td></td>
<td>• combinations: (2, 3, 4, (7, 10))</td>
</tr>
</tbody>
</table>
Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,PARAM=list addr</td>
<td>list addr: RX-type address or register (1) or (2) - (12).</td>
</tr>
<tr>
<td>,MF=(E,ctrl addr)</td>
<td>ctrl addr: RX-type address, or register (1) or (2) - (12).</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained under the standard form of ESPIE SET with the following exception:

,\( \text{MF}=(E,\text{ctrl addr}) \)

Specifies the execute form of the ESPIE macro.

\( \text{ctrl addr} \) specifies the area that the system uses to store the parameters.

Example

Give control to a installation exit routine for interruption types 1, 4, 6, 7, and 8. The exit routine address and the address of a user parameter list for the exit routine are provided in a remote control program parameter list at LIST1.

ESPIE SET,,(1,4,(6,8)),MF=(E,LIST1)

ESPIE RESET option

The RESET option of the ESPIE routine cancels the active SPIE/ESPIE environment and restores the SPIE/ESPIE environment specified by token.

Input register information

Before issuing the RESET option of the ESPIE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Token identifying the new active SPIE/ESPIE environment</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code of 0</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>
**ESPIE macro**

**14-15** Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Syntax**

The RESET option of the ESPIE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ESPIE.</td>
</tr>
<tr>
<td>ESPIE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ESPIE.</td>
</tr>
<tr>
<td>RESET</td>
<td></td>
</tr>
<tr>
<td>,token</td>
<td>token: RX-type address or register (1) or (2) - (12).</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

**RESET**

Indicates that the current ESPIE environment is to be deleted and the previously active SPIE/ESPIE environment specified by *token* is to be reestablished.

*,token*

Specifies a fullword that contains a token representing the previously active SPIE/ESPIE environment. This is the same token that ESPIE processing returned to the caller when the ESPIE trap was established using the SET option of the ESPIE macro.

If the token is zero, all SPIEs and ESPIEs are deleted.

**Return and reason codes**

None.

**Example**

Cancel the current SPIE/ESPIE environment and restore the SPIE/ESPIE environment represented by the contents of TOKEN.

ESPIE RESET, TOKEN
ESPIE TEST option

The TEST option of the ESPIE macro determines the active SPIE/ESPIE environment and returns the information in a 4-byte parameter list.

Input register information

Before issuing the TEST option of the ESPIE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Syntax

The TEST option of the ESPIE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ESPIE.</td>
</tr>
<tr>
<td>ESPIE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ESPIE.</td>
</tr>
<tr>
<td>TEST</td>
<td></td>
</tr>
</tbody>
</table>
ESPIE macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parm addr</td>
<td>parm addr: RX-type address, or register (1) or (2) - (12).</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

**TEST**

Indicates a request for information concerning the active or current
SPIE/ESPIE environment. ESPIE processing returns this information to the
caller in a 4-word parameter list located at parm addr.

`parm addr`

Specifies the address of a 4-word parameter list aligned on a fullword
boundary. The parameter list has the following form:

<table>
<thead>
<tr>
<th>Word</th>
<th>Content</th>
</tr>
</thead>
</table>
| 0    | 31-bit address of the exit routine (For 24-bit routines, the high order bit
      is set to 0. For 31-bit routines, the high order bit is set to 1.) |
| 1    | Address of the user-defined parameter list |
| 2    | Mask of program interruption types |
| 3    | Zero |

Return and reason codes

ESPIE TEST returns status information about the current ESPIE environment in
GPR 15. When control returns from ESPIE TEST, GPR 15 contains one of the
following hexadecimal return codes.

**Note**: These return codes are informational; no actions are required.

*Table 11. Return Codes for the ESPIE TEST Macro*

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 00                      | **Meaning**: An ESPIE exit is active and the 4-word parameter list contains the
      information specified in the description of the parm addr parameter. |
| 04                      | **Meaning**: A SPIE exit is active. Word 1 of the parameter list described under
      parm addr contains the address of the current PICA. Words 0, 2, and 3 of the
      parameter list contain no relevant information. |
| 08                      | **Meaning**: No SPIE or ESPIE is active. The contents of the 4-word parameter list
      contain no relevant information. |

Example

Identify the active SPIE/ESPIE environment. Return the information about the exit
routine in the 4-word parameter list, PARMLIST. Also return, in register 15, an
indication of whether a SPIE, ESPIE, or neither is active.

ESPIE TEST,PARMLIST
Chapter 6. ESTAE and ESTAEX — Specify task abnormal exit extended

Description

The ESTAE macro provides recovery capability facilities. Issuing the ESTAE macro allows the caller to intercept errors. Control is given to a caller-specified exit routine (called a recovery routine) in which the caller can perform various tasks, including diagnosing the cause of the error and specifying a retry address to avoid abnormal ending.

ESTAE type considerations: The type of ESTAE routine, that is, ESTAE or ESTAEX affects the AMODE of the recovery routine as follows. For recovery routines defined through the:

• ESTAE macro, at the time of entry to the recovery routine, the AMODE will be the same as at the time of invocation of the macro.
• ESTAEX macro, the AMODE will be the same as at the time of invocation of the macro, unless the macro was invoked in AMODE 24 in which case the recovery routine AMODE will be 31-bit.
• The AMODE at the retry point will be the same as the AMODE on entry to the recovery routine.

Various mode considerations: Depending on address space, cross-memory (the primary, secondary, and home address spaces are the same), and access register (AR) modes, you need to select the proper ESTAE type as follows:

• If your program is to execute in 31-bit addressing mode, you must use the SP Version 2 of the ESTAE macro or a later version.
• Callers that are in primary address space control (ASC) mode and not in cross-memory mode can issue either ESTAE or ESTAEX.
• Callers that are in access register (AR) mode or in cross-memory mode must use ESTAEX.
• IBM recommends that all callers use the ESTAEX macro, unless your program and your recovery routine are in 24-bit addressing mode, in which case you need to use ESTAE.

Depending on whether you code ESTAE or ESTAEX, the system passes the address of the user-specified parameter list differently. The SDWAPARM field in the SDWA contains either the address of the parameter list (ESTAE), or the address of a doubleword that contains the address and ALET of the parameter list (ESTAEX). When you run in AMODE 64 (as indicated by specifying AMODE64=YES through the SYSSTATE macro) and invoke ESTAEX, your ESTAEX routine will get control in AMODE 64. The 8-byte area pointed to by the SDWAPARM field will be the 8-byte address of the parameter area.

See the information on providing recovery in z/OS MVS Programming: Authorized Assembler Services Guide for information about writing recovery routines.

The descriptions of ESTAE and ESTAEX are:
ESTAE and ESTAEX macros

- The standard form of the ESTAE macro, which includes general information about the ESTAE and ESTAEX macros, with some specific information about the ESTAE macro. The syntax of the ESTAE macro is presented, and all ESTAE parameters are explained.
- The standard form of the ESTAEX macro, which includes information specific to the ESTAEX macro. The syntax of the ESTAEX macro is presented.
- The list form of the ESTAE and ESTAEX macros.
- The execute form of the ESTAE and ESTAEX macros.

**Note:** The ESTAE and ESTAEX macros have the same environment specifications, register information, programming requirements, restrictions and limitations, and performance implications described as follows, except where noted in the explanation for ESTAEX.

**Environment**

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state and any PSW key. To use the CANCEL, BRANCH, KEY, TOKEN, or SPIEOVERRIDE parameters, one of the following:</td>
</tr>
<tr>
<td></td>
<td>Supervisor state</td>
</tr>
<tr>
<td></td>
<td>PKM allowing key 0-7 (for BRANCH, key 0 only)</td>
</tr>
<tr>
<td></td>
<td>APF-authorized</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space</td>
</tr>
</tbody>
</table>

**Programming requirements**

If the program is in AR mode, you must use ESTAEX rather than ESTAE; issue the SYSSTATE macro with the ASCENV=AR parameter before you issue ESTAEX. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.

**Restrictions**

For Branch-entry, IBM recommends that you have no EUT FRRs.

IBM recommends that you do not use the ESTAE or ESTAEX macro to deactivate and no longer define a FESTAE recovery routine that was defined and activated by a FESTAE macro.

**Input register information**

Before issuing the ESTAE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

**Output register information**

When control returns to the caller, the general purpose registers (GPRs) contain:
ESTAE and ESTAEX macros

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3-13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
</tr>
<tr>
<td>2-13</td>
</tr>
<tr>
<td>14-15</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the ESTAE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ESTAE.</td>
</tr>
<tr>
<td>ESTAE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>One or more blanks must follow ESTAE.</td>
</tr>
<tr>
<td>exit addr</td>
<td>exit addr: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>()</td>
<td></td>
</tr>
<tr>
<td>,CT</td>
<td>Default: CT</td>
</tr>
<tr>
<td>,OV</td>
<td></td>
</tr>
</tbody>
</table>
## ESTAE and ESTAEX macros

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PARAM=list addr</code></td>
<td><code>list addr</code>: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td><code>XCTL=NO</code></td>
<td>Default: <code>XCTL=NO</code></td>
</tr>
<tr>
<td><code>XCTL=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>PURGE=NONE</code></td>
<td>Default: <code>PURGE=NONE</code></td>
</tr>
<tr>
<td><code>PURGE=QUIESCE</code></td>
<td></td>
</tr>
<tr>
<td><code>PURGE=HALT</code></td>
<td></td>
</tr>
<tr>
<td><code>ASYNCH=YES</code></td>
<td>Default: <code>ASYNCH=YES</code></td>
</tr>
<tr>
<td><code>ASYNCH=NO</code></td>
<td></td>
</tr>
<tr>
<td><code>CANCEL=YES</code></td>
<td>Default: <code>CANCEL=YES</code></td>
</tr>
<tr>
<td><code>CANCEL=NO</code></td>
<td></td>
</tr>
<tr>
<td><code>TERM=YES</code></td>
<td>Default: <code>TERM=NO</code></td>
</tr>
<tr>
<td><code>TERM=NO</code></td>
<td></td>
</tr>
<tr>
<td><code>BRANCH=NO</code></td>
<td>Default: <code>BRANCH=NO</code></td>
</tr>
<tr>
<td><code>BRANCH=YES,SVEAREA=save addr</code></td>
<td><code>save addr</code>: A-type address, or register (2) - (12) or (13).</td>
</tr>
<tr>
<td><code>KEY=SAVE</code></td>
<td><code>storage key</code>: Any numeral in the range 0-15.</td>
</tr>
<tr>
<td><code>KEY=storage key</code></td>
<td></td>
</tr>
<tr>
<td><code>RECORD=NO</code></td>
<td>Default: <code>RECORD=NO</code></td>
</tr>
<tr>
<td><code>RECORD=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>TOKEN=token addr</code></td>
<td><code>token addr</code>: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td><code>RELATED=value</code></td>
<td><code>value</code>: Any valid macro keyword specification.</td>
</tr>
<tr>
<td><code>SDWALLOC31=NO</code></td>
<td>Default: <code>SDWALLOC31=NO</code></td>
</tr>
<tr>
<td><code>SDWALLOC31=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>SPIEVERRIDE=NO</code></td>
<td>Default: <code>SPIEVERRIDE=NO</code></td>
</tr>
<tr>
<td><code>SPIEVERRIDE=YES</code></td>
<td></td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows.

*exit addr*

0

Specifies the 31-bit address of an ESTAE recovery routine to be entered if the task issuing this macro ends abnormally. If you specify 0, the most recent ESTAE recovery routine is deactivated and no longer defined.

The ESTAEX exit always gets control in 31-bit mode, regardless of the mode in which the macro was invoked.

*,CT*

,OV

Specifies that a new ESTAE recovery routine is to be defined and activated (CT), or indicates that parameters passed in this ESTAE macro are to overlay the data contained in the previous ESTAE routine (OV).

*,PARAM=list addr*

Specifies the 31-bit address of a user-defined list containing data to be used by the ESTAE routine when it is scheduled for execution.

*,XCTL=NO*

,XCTL=YES

Specifies that the ESTAE recovery routine will be deactivated and no longer defined (NO) or will remain activated and defined (YES) if this program issues an XCTL macro.

*,PURGE=NONE*

,PURGE=QUIESCE*

,PURGE=HALT

Specifies that all outstanding requests for I/O operations are not to be saved when the ESTAE routine receives control (HALT), or that I/O processing is to be allowed to continue normally when the ESTAE routine receives control (NONE), or that all outstanding requests for I/O operations are to be saved when the ESTAE routine receives control (QUIESCE). If QUIESCE is specified, the user's retry routine can restore the outstanding I/O requests.

For PURGE=QUIESCE and PURGE=HALT, RTM requests that all I/O be purged at the task level for the current task. Be aware that the purge request involves all I/O started by the task, not just the I/O started by the program that created this recovery routine. PURGE=QUIESCE must thus be used carefully, as it may wait for I/O that was not started by the program that created this recovery routine. Likewise, PURGE=HALT must be used carefully as it may terminate I/O that was not started by the program that created this recovery routine.

PURGE=NONE specifies that all control blocks affected by input/output processing can continue to change during ESTAE routine processing. If you specify PURGE=NONE and the error was an error in input/output processing, recursion develops when an input/output interruption occurs, even if the ESTAE routine is in progress. Thus, it will appear that the ESTAE routine failed when, in reality, input/output processing caused the failure.

Note:

1. You need to understand PURGE processing before using this parameter. For information about PURGE processing, see [z/OS DFSMSdfp Advanced Services](https://www.ibm.com/support/knowledgecenter/SSEQ27_5.1.0/com.ibm.zos.mksr510.doc/zos_dfsmsdftm/dfsmsdfp_adv_serv.html)
ESTAE and ESTAEX macros

2. When using PURGE, you need to consider any access-method ramifications. See the appropriate DFP information for the particular access method you are using to determine these ramifications.

3. The system performs the requested I/O processing only for the first ESTAE-type recovery routine that gets control. Subsequent routines that get control receive an indication of the I/O processing previously done, but no additional processing is performed.

\[ \text{ASYNCH=\text{YES}} \]
\[ \text{ASYNCH=\text{NO}} \]

Specifies that asynchronous exit processing will be allowed (YES) or prohibited (NO) while the user's ESTAE routine is running.

ASYNCH=\text{YES} must be coded if:
- Any supervisor services that require asynchronous interruptions to complete their normal processing are going to be requested by the ESTAE routine.
- PURGE=QUIESCE is specified for any access method that requires asynchronous interruptions to complete normal input/output processing.
- PURGE=NONE is specified and the ESTAE routine issues the CHECK macro for any access method that requires asynchronous interruptions to complete normal input/output processing.

\textbf{Note:} If ASYNCH=\text{YES} is specified and the error was an error in asynchronous exit handling, recursion will develop when an asynchronous exit handling was the cause of the failure.

\[ \text{CANCEL=\text{YES}} \]
\[ \text{CANCEL=\text{NO}} \]

Specifies whether you want to allow the recovery routine to be interrupted by cancel or detach processing.

To allow a recovery routine to be interrupted, specify CANCEL=\text{YES}.

To prevent a recovery routine from being interrupted, specify CANCEL=\text{NO}. If a cancel or detach is attempted against a recovery routine for which you have specified CANCEL=\text{NO}, MVS defers cancel and detach processing until the recovery routine returns control to the system.

\textbf{Note:}
1. If a recovery routine that runs under the CANCEL=\text{NO} option can be called by an unauthorized program running under the same task, IBM recommends that you specify ASYNCH=\text{NO} for each ESTAE(X) macro that the recovery routine issues. This also includes any ESTAE(X) macros issued by programs that the recovery routine calls.
2. If a recovery routine running under the CANCEL=\text{NO} option calls an unauthorized program, cancel and detach processing is also deferred for the called program.

\[ \text{TERM=\text{NO}} \]
\[ \text{TERM=\text{YES}} \]

Specifies that the ESTAE routine will be scheduled (YES) or will not be scheduled (NO) in the following situations:
- System-initiated logoff
- Job step timer expiration
- Wait time limit for job step exceeded
ESTAE and ESTAEX macros

- DETACH macro without the STAE=YES parameter issued from a higher-level task (possibly by the system if the higher-level task encountered an error)
- Operator cancel
- Error on a higher level task
- Error in the job step task when a nonjob step task issued the ABEND macro with the STEP parameter.
- z/OS UNIX is canceled and the user's task is in a wait in the z/OS UNIX kernel.

When the ESTAE routine is entered because of one of the preceding reasons, retry is not permitted. If a dump is requested at the time the ABEND macro is issued, it is taken before entry into the ESTAE routine.

**Note:** If DETACH was issued with the STAE parameter, the following occurs for the task to be detached:
- All ESTAE routines are entered.
- The most recently activated STAE routine is entered.
- All STAI/ESTAI routines are entered unless one of the STAI routines issues return code 16.

In these cases, entry to the routine occurs before dumping and retry is not permitted.

**BRANCH=NO**
**BRANCH=YES, SVEAREA=save addr**

Specifies that an SVC entry to the ESTAE service routine is to be performed (NO) or that a branch entry is to be performed (YES). The save area is a 72-byte area used to save the general registers. If the caller is not in key zero, the KEY parameter must be specified.

BRANCH and SVEAREA are not valid on ESTAEX.

**KEY=SAVE**
**KEY=storage key**

Specifies that supervisor state users who are not in key zero can use the branch entry interface to the ESTAE service routine.

If the user specifies KEY=SAVE, the macro saves the current PSW protection key in register 2 and issues a set protection key instruction (SPKA) to change to protection key zero. When the ESTAE service routine returns control, it restores the original PSW key from register 2. Therefore, the user should save register 2 before the macro expansion and restore it afterwards. Specifying KEY=SAVE destroys the contents of register 2 during the macro expansion.

On the other hand, if the user knows the current PSW protection key, he may specify it directly in the form KEY=(0-15) to eliminate saving and restoring the original protection key. This procedure eliminates an IPK instruction and prevents the use of register 2 in the macro expansion.

KEY is not valid on ESTAEX. KEY is optional and valid only with BRANCH=YES, SVEAREA=save addr.

**RECORD=NO**
**RECORD=YES**

Specifies whether the system diagnostic work area (SDWA) is to be recorded in SYS1.LOGREC. If you specify RECORD=YES, the system records the entire SDWA (including the fixed length base, the variable length recording area, and the recordable extensions) in SYS1.LOGREC when the associated ESTAE
ESTAE and ESTAEX macros

recovery routine returns control, unless the recovery routine indicates otherwise by issuing the SETRP macro with RECORD=NO.

If you specify RECORD=NO, the system does not record the SDWA in SYS1.LOGREC, unless the recovery routine indicates otherwise by issuing the SETRP macro with RECORD=YES.

\texttt{\textbf{,TOKEN=token addr}}

Specifies that a four-byte token is to be associated with the ESTAE routine. Unauthorized or accidental destruction of the ESTAE routine is prevented because the ESTAE cannot be canceled or overlaid unless the same token is specified.

With CT: ESTAE processing places the token created for this request in the location specified by \textit{token addr} as well as in the ESTAE parameter list.

With OV: ESTAE processing locates the specified ESTAE routine for the current RB and replaces the routine information. If there are any newer ESTAE routines for the RB, they are deactivated and no longer defined.

With a recovery routine address of 0: ESTAE processing locates the specified ESTAE routine for the current RB and deactivates the routine. The routine is no longer defined. Any newer ESTAE routines for the RB are deactivated and no longer defined.

\texttt{\textbf{,RELATED=value}}

Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and content of the information specified are at the discretion of the user, and may be any valid coding values.

\texttt{\textbf{,SDWALOC31=NO}}, \texttt{\textbf{,SDWALOC31=YES}}

Specifies that the SDWA be in 31-bit storage (YES) or the default 24-bit storage (NO). You must specify SDWALOC31=YES when the your program is running in AMODE 31 and you are using 64-bit general purpose registers, because the time-of-error 64-bit GPRs are only presented to routines with an SDWA in 31-bit storage. Only routines with an SDWA in 31-bit storage can retry while setting those registers.

\textbf{Note:} The SDWALOC31= parameter applies to ESTAE only. (For ESTAEX, the SDWA is always in 31-bit storage.)

\texttt{\textbf{,SPIEOVERRIDE=NO}}, \texttt{\textbf{,SPIEOVERRIDE=YES}}

SPIEOVERRIDE specifies that the ESTAEX recovery exit must receive control for all program exceptions even if a SPIE or ESPIE exit is established.

While the recovery routine that requests this parameter is established, no SPIE or ESPIE exit can receive control.

You can use this parameter to ensure that the ESTAEX recovery exit receives control for all program exceptions that occur while running in Problem state.

The SPIEOVERRIDE parameter is not required for programs that run in Supervisor state, run in cross-memory, or hold any lock, because SPIE and ESPIE exits are not eligible to receive control in these environments.

SPIEOVERRIDE is not valid on ESTAE.

The default value is SPIEOVERRIDE=NO.
ABEND codes

None.

Return and reason codes

When control returns to the instruction following the ESTAE macro, GPR 15 contains one of the following return codes and GPR 0 contains one of the following reason codes.

Table 12. Return and Reason Codes for the ESTAE Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None</td>
<td><strong>Meaning:</strong> Successful completion of the ESTAE request. <strong>Action:</strong> None.</td>
</tr>
<tr>
<td>04</td>
<td>00</td>
<td><strong>Meaning:</strong> Program error. ESTAE OV was specified but ESTAE CT was performed. No valid ESTAE recovery routine existed.</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td><strong>Meaning:</strong> Program error. ESTAE OV was specified but ESTAE CT was performed. The last ESTAE recovery routine was not owned by the user's RB. <strong>Action:</strong> Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate.</td>
</tr>
<tr>
<td>04</td>
<td>08</td>
<td><strong>Meaning:</strong> Program error. ESTAE OV was specified but ESTAE CT was performed. The last ESTAE recovery routine was not created at the current linkage stack level. <strong>Action:</strong> Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate.</td>
</tr>
<tr>
<td>04</td>
<td>0C</td>
<td><strong>Meaning:</strong> Program error. ESTAE OV was specified but ESTAE CT was performed. The last recovery routine was not an ESTAE recovery routine. <strong>Action:</strong> Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate.</td>
</tr>
</tbody>
</table>
| 0C                      | None                    | **Meaning:** Program error. A recovery routine address equal to zero was specified, and either  
  • There are no recovery routines for this TCB,  
  • The most recent recovery routine is not owned by the caller,  
  • The most recent recovery routine is not an ESTAE recovery routine, or  
  • The ESTAE was created with the TOKEN parameter and on a deactivate request, either  
    – The token was not specified or  
    – The token does not match. **Action:** Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate. |
| 10                      | None                    | **Meaning:** System error. An unexpected error was encountered while this request was being processed. **Action:** Rerun your program one or more times. If the problem persists, record the return and reason codes and supply them to the appropriate IBM support personnel. |
| 14                      | None                    | **Meaning:** Environmental error. ESTAE was unable to obtain storage for a system data area. **Action:** Free some storage and reissue the ESTAE macro. |
### ESTAE and ESTAEX Macros

#### Table 12. Return and Reason Codes for the ESTAE Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 18                      | None                    | **Meaning:** Program error. ESTAE OV request was invalid for one of the following reasons:  
|                         |                         | • ESTAE OV with the TOKEN parameter was specified but  
|                         |                         | – No ESTAE recovery routine exists or  
|                         |                         | – The recovery routine is not an ESTAE recovery routine  
|                         |                         | created with the matching token value by the current  
|                         |                         | RB.  
|                         |                         | • ESTAE OV without the TOKEN parameter was specified  
|                         |                         | but the ESTAE recovery routine was created with the  
|                         |                         | TOKEN parameter.  
|                         |                         | **Action:** Correct the environment and either reissue the  
|                         |                         | ESTAE macro or rerun your program, as appropriate. |
| 1C                      | None                    | **Meaning:** Program error. ESTAE was unable to access the input parameter list.  
|                         |                         | **Action:** Make sure the parameter list is in the primary address space and reissue the ESTAE macro. |
| 20                      | None                    | **Meaning:** Program error. XCTL=YES was rejected because the linkage stack was not at the same level as it was when the RB was created.  
|                         |                         | **Action:** Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate. |
| 24                      | None                    | **Meaning:** Program error. A recovery routine address equal to zero was specified, but it was rejected because no ESTAE recovery routines were active for the current linkage stack level.  
|                         |                         | **Action:** Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate. |
| 28                      | None                    | **Meaning:** Program error. ESTAE OV was specified, but it was rejected because no ESTAE recovery routines were active for the current linkage stack level.  
|                         |                         | **Action:** Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate. |
| 30                      | None                    | **Meaning:** Program error. Branch-entered ESTAE CT was specified, but it was rejected because the caller has a cross-memory environment.  
|                         |                         | **Action:** Use ESTAEX for programs that run in a cross-memory environment. |

#### Example 1

If an error occurs, pass control to the ESTAE routine specified by register 4, allow asynchronous exit processing, do not allow special error processing, do not branch enter, and default to CT and PURGE=None.

**ESTAE (4), ASYNCH=YES, TERM=NO, BRANCH=NO**

#### Example 2

If an error occurs, pass control to the ESTAE routine specified by register 4. The address of the ESTAE parameter list is in register 2. Place the token associated with this ESTAE routine in TOKENFLD.

**ESTAE (4), PARAM=(2), TOKEN=TOKENFLD**
Example 3

If an error occurs, pass control to the ESTAE routine labeled ADDR, allow synchronous exit processing, halt I/O, allow special error processing, branch enter, use the 72-byte save area at SADDR, and execute the execute form of the macro. EXEC is the label of the ESTAE parameter list built by a list form of the macro elsewhere in this program.

```
ESTAE ADDR,ASYNCH=YES,PURGE=HALT,TERM=YES,BRANCH=YES,
        X
        SVEAREA=SADDR,MF=(E,EXEC)
```

Example 4

Request an overlay of the existing ESTAE recovery routine with the following options: the address of the parameter list is at PLIST, I/O will be halted, no asynchronous exits will be taken, ownership will be transferred to the new request block resulting from any XCTL macros.

```
ESTAE ADDR,OV,PARAM=PLIST,XCTL=YES,PURGE=HALT,ASYNCH=NO
```

Example 5

Provide the pointer to the recovery code in the register called EXITPTR, place the address of the ESTAE parameter list in register 9. Register 8 points to the area where the ESTAE parameter list (created with the MF=L option) was moved.

```
ESTAE (EXITPTR),PARAM=(9),MF=(E,(8))
```

ESTAEX - Specify task abnormal exit extended

**Note:** The ESTAEX macro has the same environment specifications, register information, programming requirements, restrictions and limitations, and performance implications as the ESTAE macro, with the exceptions that follow.

**Environment**

The requirements for the caller of ESTAEX that are different from ESTAE are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
</tbody>
</table>

**Programming requirements**

If the program is in AR mode:

- Issue the SYSSTATE macro with the ASCENV=AR parameter before you issue ESTAEX. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.
- User parameters, specified on the PARAM parameter, can be located in any address space.

**Restrictions**

The caller of ESTAEX cannot have an EUT FRR established.

The parameters on the standard form of the ESTAEX macro are the same as for the standard form of the ESTAE macro, except BRANCH, SVEAREA, and KEY, which are not valid for ESTAEX.
## Syntax

The standard form of the ESTAEX macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: Symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede ESTAEX.</td>
</tr>
<tr>
<td>ESTAEX</td>
<td>One or more blanks must follow ESTAEX.</td>
</tr>
<tr>
<td><code>exit addr</code></td>
<td><code>exit addr</code>: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td><code>0</code></td>
<td></td>
</tr>
<tr>
<td>,CT</td>
<td>Default: CT</td>
</tr>
<tr>
<td>,OV</td>
<td></td>
</tr>
<tr>
<td>,PARAM=list addr</td>
<td><code>list addr</code>: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,XCTL=NO</td>
<td>Default: XCTL=NO</td>
</tr>
<tr>
<td>,XCTL=YES</td>
<td></td>
</tr>
<tr>
<td>,PURGE=NONE</td>
<td>Default: PURGE=NONE</td>
</tr>
<tr>
<td>,PURGE=QUIESCE</td>
<td></td>
</tr>
<tr>
<td>,PURGE=HALT</td>
<td></td>
</tr>
<tr>
<td>,ASYNCH=YES</td>
<td>Default: ASYNCH=YES</td>
</tr>
<tr>
<td>,ASYNCH=NO</td>
<td></td>
</tr>
<tr>
<td>,CANCEL=YES</td>
<td>Default: CANCEL=YES</td>
</tr>
<tr>
<td>,CANCEL=NO</td>
<td></td>
</tr>
<tr>
<td>,TERM=NO</td>
<td>Default: TERM=NO</td>
</tr>
<tr>
<td>,TERM=YES</td>
<td></td>
</tr>
<tr>
<td>,RECORD=NO</td>
<td>Default: RECORD=NO</td>
</tr>
<tr>
<td>,RECORD=YES</td>
<td></td>
</tr>
<tr>
<td>,TOKEN=token addr</td>
<td><code>token addr</code>: A-type address, or register (2) - (12).</td>
</tr>
</tbody>
</table>
**ESTAE and ESTAEX macros**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,RELATED=value</td>
<td>value: Any valid macro keyword specification.</td>
</tr>
<tr>
<td>,SPIEOVERRIDE=NO</td>
<td><strong>Default</strong>: SPIEOVERRIDE=NO</td>
</tr>
<tr>
<td>,SPIEOVERRIDE=YES</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained under the syntax for the standard form of the ESTAE macro.

**ABEND codes**

None.

**Return and reason codes**

When control returns to the instruction following the ESTAEX macro, the return code in GPR 15 and the reason code in GPR 0 might be different from those for the ESTAE macro. The following table lists the return and reason codes for ESTAEX.

*Table 13. Return and Reason Codes for the ESTAEX Macro*

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None</td>
<td><strong>Meaning</strong>: Successful completion of the ESTAEX request. <strong>Action</strong>: None.</td>
</tr>
<tr>
<td>04</td>
<td>00</td>
<td><strong>Meaning</strong>: Program error. ESTAEX OV was specified but ESTAEX CT was performed. No valid ESTAE recovery routine existed. <strong>Action</strong>: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td><strong>Meaning</strong>: Program error. ESTAEX OV was specified but ESTAEX CT was performed. The last ESTAE recovery routine was not owned by the user's RB. <strong>Action</strong>: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.</td>
</tr>
<tr>
<td>04</td>
<td>08</td>
<td><strong>Meaning</strong>: Program error. ESTAEX OV was specified but ESTAEX CT was performed. The last ESTAE recovery routine was not created at the current linkage stack level. <strong>Action</strong>: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.</td>
</tr>
<tr>
<td>04</td>
<td>0C</td>
<td><strong>Meaning</strong>: Program error. ESTAEX OV was specified but ESTAEX CT was performed. The last recovery routine was not an ESTAE recovery routine. <strong>Action</strong>: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.</td>
</tr>
<tr>
<td>08</td>
<td>None</td>
<td><strong>Meaning</strong>: Program error. The ESTAEX request was not valid. <strong>Action</strong>: Correct the request and either reissue the ESTAEX macro or rerun your program, as appropriate.</td>
</tr>
</tbody>
</table>
### ESTAE and ESTAEX macros

Table 13. Return and Reason Codes for the ESTAEX Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0C                      | None                    | **Meaning:** Program error. A recovery routine address equal to zero was specified, and either  
  • There are no recovery routines for this TCB,  
  • The most recent recovery routine is not owned by the caller,  
  • The most recent recovery routine is not an ESTAE recovery routine, or  
  • The ESTAE was created with the TOKEN parameter and on a deactivate request, either  
    – The token was not specified or  
    – The token does not match.  
  **Action:** Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate. |
| 10                      | None                    | **Meaning:** System error. An unexpected error was encountered while this request was being processed.  
  **Action:** Rerun your program one or more times. If the problem persists, record the return and reason codes and supply them to the appropriate IBM support personnel. |
| 14                      | None                    | **Meaning:** Environmental error. ESTAEX was unable to obtain storage for a system data area.  
  **Action:** Free some storage and reissue the ESTAEX macro. |
| 18                      | None                    | **Meaning:** Program error. ESTAEX OV was requested and one of the following occurred:  
  • The TOKEN parameter was specified and the ESTAE recovery routine is not owned by the current RB  
  • The TOKEN parameter was not specified but the ESTAE recovery routine was created with the TOKEN parameter.  
  **Action:** Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate. |
| 1C                      | None                    | **Meaning:** Program error. ESTAEX was unable to access the input parameter list.  
  **Action:** Make sure the parameter list is contained in the primary address space and reissue the ESTAEX macro. |
| 20                      | None                    | **Meaning:** Program error. XCTL=YES was rejected because the linkage stack was not at the same level as it was when the RB was created.  
  **Action:** Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate. |
| 24                      | None                    | **Meaning:** Program error. A recovery routine address equal to zero was specified, but it was rejected because no ESTAE recovery routines were active for the current linkage stack level.  
  **Action:** Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate. |
| 28                      | None                    | **Meaning:** Program error. The caller was disabled.  
  **Action:** Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate. |
### ESTAE and ESTAEX macros

#### Table 13. Return and Reason Codes for the ESTAEEX Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 2C                      | None                    | **Meaning**: Program error. The caller was locked.  
**Action**: Correct the environment and either reissue the ESTAEEX macro or rerun your program, as appropriate. |
| 30                      | None                    | **Meaning**: Program error. The caller had FRRs on the current FRR stack.  
**Action**: Correct the environment and either reissue the ESTAEEX macro or rerun your program, as appropriate. |
| 34                      | None                    | **Meaning**: Program error. The caller was in SRB mode.  
**Action**: Correct the environment and either reissue the ESTAEEX macro or rerun your program, as appropriate. |

### Example

The following example show how to establish an ESTAEEX recovery routine that receives control for all abends, including CANCEL or DETACH abends, and overrides any SPIE or ESPIE exit that is established:

```
ESTAEEX addr,PARM=parmaddr,TERM=YES,SPIEVERRIDE=YES
```

### ESTAE and ESTAEX - List form

The list form of ESTAE or ESTAEX is used to construct a remote control parameter list.

#### Syntax

The list form of ESTAE or ESTAEX is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: Symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ESTAE or ESTAEX.</td>
</tr>
<tr>
<td>ESTAE</td>
<td></td>
</tr>
<tr>
<td>ESTAEX</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ESTAE or ESTAEX.</td>
</tr>
<tr>
<td><code>exit addr</code></td>
<td><code>exit addr</code>: A-type address.</td>
</tr>
<tr>
<td><code>,PARAM=list addr</code></td>
<td><code>list addr</code>: A-type address.</td>
</tr>
<tr>
<td><code>,PURGE=NONE</code></td>
<td><strong>Default</strong>: PURGE=NONE</td>
</tr>
<tr>
<td><code>,PURGE=QUIESCE</code></td>
<td></td>
</tr>
<tr>
<td><code>,PURGE=HALT</code></td>
<td></td>
</tr>
</tbody>
</table>
## ESTAE and ESTAEX macros

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,ASYNCH=YES</td>
<td>Default: ASYNCH=YES</td>
</tr>
<tr>
<td>,ASYNCH=NO</td>
<td></td>
</tr>
<tr>
<td>,CANCEL=YES</td>
<td>Default: CANCEL=YES</td>
</tr>
<tr>
<td>,CANCEL=NO</td>
<td></td>
</tr>
<tr>
<td>,TERM=NO</td>
<td>Default: TERM=NO</td>
</tr>
<tr>
<td>,TERM=YES</td>
<td></td>
</tr>
<tr>
<td>,RECORD=NO</td>
<td>Default: RECORD=NO</td>
</tr>
<tr>
<td>,RECORD=YES</td>
<td></td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>value: Any valid macro keyword specification.</td>
</tr>
<tr>
<td>,SDWALOC31=NO</td>
<td>Default: SDWALOC31=NO</td>
</tr>
<tr>
<td>,SDWALOC31=YES</td>
<td>Note: SDWALOC31 is supported only by ESTAE.</td>
</tr>
<tr>
<td>,SPIEOVERRIDE=NO</td>
<td>Default: SPIEOVERRIDE=NO</td>
</tr>
<tr>
<td>,SPIEOVERRIDE=YES</td>
<td></td>
</tr>
<tr>
<td>,MF=L</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the ESTAE or ESTAEX macro with the following exception:

- **,MF=L**
  - Specifies the list form of the ESTAE or ESTAEX macro.

### ESTAE or ESTAEX - Execute form

A remote control parameter list is used in, and can be modified by, the execute form of the ESTAE or ESTAEX macro. The control parameter list can be generated by the list form of the ESTAE or ESTAEX macro. Any combination of exit addr, PARAM, XCTL, PURGE, ASYNCH, TERM, RECORD, TOKEN, and SPIEOVERRIDE can be specified to dynamically change the contents of the remote ESTAE or ESTAEX parameter list. If the TOKEN parameter was previously specified and is to be used again without change, TKNPASS=YES must be coded. Any fields not specified on the macro remain as they were before the current ESTAE or ESTAEX request was made.

**Note:** To ensure that the ESTAE or ESTAEX parameters are correct, the control parameter list specified for the execute form of the ESTAE and ESTAEX macros must be initialized from a list form of the macro.
**Syntax**

The execute form of the ESTAE or ESTAEX macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: Symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede ESTAE or ESTAEX.</td>
</tr>
<tr>
<td>ESTAE</td>
<td></td>
</tr>
<tr>
<td>ESTAEX</td>
<td></td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must follow ESTAE or ESTAEX.</td>
</tr>
<tr>
<td><code>exit addr</code></td>
<td><code>exit addr</code>: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td><code>0</code></td>
<td></td>
</tr>
<tr>
<td><code>,CT</code></td>
<td></td>
</tr>
<tr>
<td><code>,OV</code></td>
<td></td>
</tr>
<tr>
<td><code>,PARAM=list addr</code></td>
<td><code>list addr</code>: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td><code>,XCTL=NO</code></td>
<td></td>
</tr>
<tr>
<td><code>,XCTL=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>,PURGE=None</code></td>
<td></td>
</tr>
<tr>
<td><code>,PURGE=QUIESCE</code></td>
<td></td>
</tr>
<tr>
<td><code>,PURGE=HALT</code></td>
<td></td>
</tr>
<tr>
<td><code>,ASYNCH=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>,ASYNCH=NO</code></td>
<td></td>
</tr>
<tr>
<td><code>,CANCEL=YES</code></td>
<td><strong>Default</strong>: CANCEL=YES</td>
</tr>
<tr>
<td><code>,CANCEL=NO</code></td>
<td></td>
</tr>
<tr>
<td><code>,TERM=NO</code></td>
<td></td>
</tr>
<tr>
<td><code>,TERM=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>,BRANCH=NO</code></td>
<td><strong>Note</strong>: BRANCH and SVEAREA are not valid on ESTAEX.</td>
</tr>
<tr>
<td><code>,BRANCH=YES, SVEAREA=save</code></td>
<td><code>save addr</code>: RX-type address, or register (2) - (12) or (13).</td>
</tr>
<tr>
<td><code>addr</code></td>
<td></td>
</tr>
<tr>
<td><code>,KEY=SAVE</code></td>
<td><code>storage key</code>: Any numeral in the range 0-15.</td>
</tr>
<tr>
<td><code>,KEY=storage key</code></td>
<td><strong>Note</strong>: KEY is not valid on ESTAEX.</td>
</tr>
</tbody>
</table>
ESTAE and ESTAEX macros

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,RECORD=NO</td>
<td></td>
</tr>
<tr>
<td>,RECORD=YES</td>
<td></td>
</tr>
<tr>
<td>,TOKEN=token addr</td>
<td>token addr: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,TKNPASS=NO</td>
<td>Default: TKNPASS=NO</td>
</tr>
<tr>
<td>,TKNPASS=YES</td>
<td></td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>value: Any valid macro keyword specification.</td>
</tr>
<tr>
<td>,SDWALOC31=NO</td>
<td>Default: SDWALOC31=NO</td>
</tr>
<tr>
<td>,SDWALOC31=YES</td>
<td>Note: SDWALOC31 is supported only by ESTAE.</td>
</tr>
<tr>
<td>,SPIEOVERRIDE=NO</td>
<td>Default: SPIEOVERRIDE=NO</td>
</tr>
<tr>
<td>,SPIEOVERRIDE=YES</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,ctrl addr)</td>
<td>ctrl addr: RX-type address, or register (1) or (2) - (12).</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained under the standard form of the ESTAE or ESTAEX macro, with the following exceptions:

,TKNPASS=NO
,TKNPASS=YES

Specifies that a previously-specified token, indicated in the parameter list, should be ignored (NO), or should remain part of the specification (YES).

,MF=(E,ctrl addr)

Specifies the execute form of the ESTAE or ESTAEX macro using a remote control parameter list.
Chapter 7. ETCON — Connect entry table

Description
The ETCON macro connects one or more previously created entry tables to the specified linkage table indexes in the current home address space. If an entry table is connected to a system linkage index (an index reserved with the SYSTEM=YES option of the LXRES macro), the entry table is connected to the linkage table of every address space, both present and future.

The connection created by the ETCON macro remains in effect until one of the following occurs:
- The ETDIS macro removes the connection.
- The entry table owner terminates.
- The address space to which the table is connected terminates unless the connection was to a system linkage index.
- The system is re-IPLed.

Related macros
ETDEF, ETCRE, ETDES, and ETDIS

Environment
The requirements for callers of ETCON are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PKM 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt Status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>The parameter list passed to the ETCON macro must be addressable in primary mode at the time the macro is issued.</td>
</tr>
</tbody>
</table>

Programming requirements
None.

Restrictions
The restrictions on the use of the ETCON macro are the following:
- If an entry table contains entries that cause address space switches, the entry table owner must have PT and SSAR authorization to issue PT and SSAR instructions to the home address space.
- An entry table can be connected only once to a single linkage table.
- The linkage index and the entry table being connected must be owned by the same task (the cross memory resource owning task of the home address space).
ETCON macro

Any violation of these restrictions causes the system to abnormally end the calling program.

Input register information
The ETCON macro is sensitive to the SYSSTATE macro with the OSREL=ZOSV1R6 parameter:
- If the caller has issued the SYSSTATE macro with the OSREL=ZOSV1R6 parameter (Version 1 Release 6 of z/OS or later) before issuing the ETCON macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.
- Otherwise, the caller must ensure that the following general purpose register contains the specified information:
  
<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of an 18-word save area</td>
</tr>
</tbody>
</table>

Output register information
After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When using the standard form of ETCON, do not use register 2 as your program’s base register. The macro modifies register 2 and then uses a branch instruction. Register 2 is restored by the time control returns to your program.

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

Performance implications
None.

Syntax
The ETCON macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
</tbody>
</table>
ETCON macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>One or more blanks must precede ETCON.</td>
</tr>
<tr>
<td>ETCON</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ETCON.</td>
</tr>
<tr>
<td>TKLIST=addr</td>
<td>addr: RX-type address or register (0) - (12).</td>
</tr>
<tr>
<td>,LXLIST=lx list addr</td>
<td>lx list addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ELXLIST=elx list addr</td>
<td>elx list addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>value: Any valid macro keyword specification.</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

**TKLIST=addr**

Specifies the address of a list of fullword tokens representing the entry tables to be connected to the linkage table. The first entry in the list must be the number of tokens that follow (from 1 to 32). The tokens are the values returned in register 0 when the ETCRE macro is issued.

**,LXLIST=lx list addr**

`lx list addr` specifies the address of a list of linkage index (LX) values to which the specified entry tables are to be connected. The list contains fullword entries, the first of which must be the number of linkage index values that follow (from 1 to 32). The number of linkage index values must be the same as the number of tokens. The first entry table is connected to the first linkage index; the second entry table is connected to the second linkage index, and so on.

**,ELXLIST=elx list addr**

`elx list addr` specifies the address of an area that contains extended linkage index (LX) values to which the specified entry tables are to be connected. The first word in the area must be the number of extended LX values that follow (from 1 to 32). Each subsequent eight bytes contains an extended LX value which consists of a 4-byte sequence number followed by an LX value. The number of extended linkage index values must be the same as the number of tokens. The first entry table is connected to the first linkage index; the second entry table is connected to the second linkage index, and so on. If the sequence number in the entry is incorrect, the system issues abend X'052' with reason code X'051B'.

**,RELATED=value**

Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user and can be any valid coding values.
ETCON macro

ABEND codes

052
053

See z/OS MVS System Codes for an explanation and programmer responses for these codes.

Return codes

When ETCON macro returns control to your program, GPR 15 contains a return code.

Table 14. Return Code for the ETCON Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td><strong>Meaning:</strong> The specified connections were successfully made.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
</tbody>
</table>

Examples

For examples of the use of this and other cross memory macros, refer to the chapter on cross memory communication in z/OS MVS Programming: Extended Addressability Guide.

ETCON - List form

The list form of the ETCON macro constructs a nonexecutable parameter list. This list, or a copy of it for reentrant programs, can be referred to by the execute form of the macro.

Syntax

The list form of the ETCON macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name:</em> Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ETCON.</td>
</tr>
<tr>
<td>ETCON</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ETCON.</td>
</tr>
<tr>
<td>TKLIST=addr</td>
<td><em>addr:</em> A-type address.</td>
</tr>
<tr>
<td>,LXLIST=addr</td>
<td><em>addr:</em> A-type address.</td>
</tr>
<tr>
<td>,RELATED=value</td>
<td><em>value:</em> Any valid macro keyword specification.</td>
</tr>
</tbody>
</table>
ETCON macro

Parameters

The parameters are explained under the standard form of the ETCON macro, with the following exception:

```
,MF=L
```

Specifies the list form of the ETCON macro.

ETCON - Execute form

The execute form of the ETCON macro can refer to and modify a remote parameter list created by the list form of the macro.

Syntax

The execute form of the ETCON macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ETCON.</td>
</tr>
<tr>
<td>TKLIST=addr</td>
<td>addr: RX-type address or register (0) - (12).</td>
</tr>
<tr>
<td>,LXLIST=addr</td>
<td>addr: RX-type address or register (0) - (12).</td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>value: Any valid macro keyword specification.</td>
</tr>
<tr>
<td>,MF=(E,cntl addr)</td>
<td>cntl addr: RX-type address or register (0) - (12).</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained under the standard form of the ETCON macro with the following exception:

```
,MF=(E,cntl addr)
```

Specifies the execute form of the ETCON macro. This form uses a remote parameter list.
ETCON macro
Chapter 8. ETCRE — Create entry table

Description

The ETCRE macro builds a program-call entry table based upon descriptions of each entry. A token representing the created entry table is returned to the requestor. You must use this token in all subsequent references to the entry table.

Related macros

ETDEF, ETDES, ETCON, and ETDIS

Environment

These are the requirements for the caller:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PKM 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN or PASN¬=HASN¬=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in primary address space</td>
</tr>
</tbody>
</table>

Programming requirements

Before issuing ETCRE, the caller must create the ETD parameter list that ETCRE uses as input. The parameter list defines the names and characteristics of the program call (PC) routines that the entry table will define. To create the parameter list, the caller can issue the ETDEF macro or can code the data constants needed to define the list. If data constants are coded, the caller can use mapping macro IHAETD to map them.

The created entry table is owned by the cross memory resource ownership task in the current home address space. When the cross memory resource ownership task terminates, entry tables are disconnected and freed.

Note: Programs written before SP/Version 3, which use data constants to define the parameter list (the resulting ETD was called a format 0 ETD) and which use IHAETD to map the data area, will still work. For information about the format 0 ETD, see z/OS MVS Data Areas in the z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/)

Restrictions

None.

Input register information

The ETCRE macro is sensitive to the SYSSTATE macro with the OSREL=ZOSV1R6 parameter

- If the caller has issued the SYSSTATE macro with the OSREL=ZOSV1R6 parameter (Version 1 Release 6 of z/OS or later) before issuing the ETCRE
ETCRE macro

macro, the caller does not have to place any information into any general
purpose register (GPR) unless using it in register notation for a particular
parameter, or using it as a base register.

- Otherwise, the caller must ensure that the following general purpose register
  contains the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of an 18-word save area</td>
</tr>
</tbody>
</table>

Output register information

After the caller issues the macro, the macro might use some registers as work
registers or might change the contents of some registers. When the macro returns
control to the caller, the contents of these registers are not the same as they were
before the macro was issued. Therefore, if the caller depends on these registers
containing the same value before and after issuing the macro, the caller must save
these registers before issuing the macro and restore them after the system returns
control.

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The 32-bit token associated with the new entry table</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the macro</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the macro</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

Performance implications

None.

Syntax

The ETCRE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ETCRE.</td>
</tr>
<tr>
<td>ETCRE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ETCRE.</td>
</tr>
<tr>
<td>ENTRIES=addr</td>
<td>addr: RX-type address or register (0) - (12).</td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>value: Any valid macro keyword specification.</td>
</tr>
</tbody>
</table>
ETCRE macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>

**Parameters**

The parameters are explained as follows:

**ENTRIES=addr**

Specifies the address of the parameter list that defines the PC routines.

An entry index value that does not have a description results in an invalid entry in the entry table. If the program name field in an ETD entry contains zeros, an invalid entry is created for that entry index. A program call to an invalid entry causes the caller to be abnormally terminated. The ETCRE caller is abnormally terminated if any of the reserved fields are nonzero or if the system cannot locate the specified program name.

**RELATED=value**

Specifies information used to self-document macros by relating functions or services to corresponding services performed elsewhere. The format and contents of the information specified can be any valid coding values.

**ABEND codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>052</td>
<td>See <a href="http://www.ibm.com">z/OS MVS System Codes</a> for an explanation and programmer responses for this code.</td>
</tr>
<tr>
<td>053</td>
<td></td>
</tr>
</tbody>
</table>

**Return codes**

When ETCRE macro returns control to your program, GPR 15 contains a return code.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 00                      | **Meaning**: The entry table is successfully created.  
**Action**: None required. |

**Example**

Show the relationship between the ETCRE and the ETDEF macros. ETDEF builds an entry table descriptor (ETD) that contains two ETD entries. The first entry, associated with PROGRAM1, is for a PC routine that runs in supervisor state. The second entry, associated with PROGRAM2, is for a PC routine that runs in problem state.

```plaintext
* CREATE THE ENTRY TABLE
*
  LA 2,ETSTART
  ETCRE ENTRIES=(2)
  .
  *
```
ETCRE macro

* DEFINE START OF ETD

* ETSTART ETDEF TYPE=INITIAL START ETD

* DEFINE ENTRIES

* ETEX2 ETDEF TYPE=ENTRY,PROGRAM='PROGRAM1',AKM=(0:15)
  ETDEF TYPE=ENTRY,PROGRAM='PROGRAM2',AKM=(0:7)

* DEFINE END OF ETD

* ETDEF TYPE=FINAL
Chapter 9. ETDEF — Create an entry table descriptor (ETD)

Description

The ETDEF macro builds and modifies the parameter that the ETCRE macro uses to build an entry table. The parameter, called the entry table descriptor (ETD), consists of a header, followed by one or more entries, called ETD entries, each one describing a PC routine. The address of the ETD is input to the ENTRIES parameter on the ETCRE macro.

The TYPE parameter on the ETDEF macro determines which process the ETDEF macro is to perform:

- ETDEF TYPE=INITIAL generates the header for the ETD. (Issue this macro once for each ETD.)
- ETDEF TYPE=ENTRY generates one ETD entry. (You can issue this macro up to 256 times for each ETD.)
- ETDEF TYPE=FINAL terminates the ETD. (Issue this macro once for each ETD.)
- ETDEF TYPE=SET,ETEADR replaces the variable fields of an existing ETD entry.
- ETDEF TYPE=SET,HEADER changes the number of entries in an existing ETD header.

Related macros

ETDES, ETCRE, ETCON, and ETDIS

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem or Supervisor state</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN or PASN≠HASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Serialization</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>None</td>
</tr>
<tr>
<td>Locks</td>
<td>None</td>
</tr>
<tr>
<td>Control parameters</td>
<td>None</td>
</tr>
</tbody>
</table>

Programming requirements

You need to create an ETD at compile time through TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL parameters and initialize the information for the entries at execution time through TYPE=SET,ETEADR. Therefore, ETDEF with the TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL parameters works like a list form of the macro. However, unlike the execute form of a macro, which changes only the values you specify, the TYPE=SET form of ETDEF completely replaces the variable fields of an ETD entry, taking the default values for any parameters you omit, and leaves constant fields as initialized. This information describes the two forms separately.
ETDEF macro

Although ETDEF is the preferred programming interface, if you have an existing ETD and you want to update the parameters (for example, change the user parameter), you might choose to use the IHAETD mapping macro instead of ETDEF. If you change an existing ETD, without using any of the function of MVS/SP Version 3, you can use IHAETD with the format number of "0". The format of IHAETD is in z/OS MVS Data Areas in the z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/) under "ETD".

Note: When changing code to use ETDEF in place of the IHAETD mapping macro, be sure to specify PC=BASIC so that the PC does not become a stacking PC. If you want to change an existing PC routine to a stacking PC, be sure to change the PT instruction in the PC routine to a PR.

Restrictions
None.

Register information
The ETDEF macro does not use any registers, except for those you use to specify parameters.

Performance implications
None.

TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL parameters
The ETDEF macro with the TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL options works like a list form of a macro.

Syntax
This form is described as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ETDEF.</td>
</tr>
<tr>
<td>ETDEF</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ETDEF.</td>
</tr>
</tbody>
</table>

**TYPE=INITIAL**

**Valid Parameters:** RELATED

**TYPE=ENTRY**

**Required Parameters:** PROGRAM or ROUTINE, AKM

EKM, ARR, ASCMODE, EAX, EK, PARM1, PARM2, PC, PKM, SASN, SSWITCH, STATE, RELATED, ASYNCH, CANCEL

**TYPE=FINAL**

RELATED

**AKM=key-list**

key-list: List of keys or key ranges where a key is a number 0 - 15.
### ETDEF macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ARR=arr</code></td>
<td><code>arr</code>: A-type address, or alphanumeric character string enclosed by single quotation marks.</td>
</tr>
<tr>
<td><code>ARRCOND=NO</code></td>
<td>Default: <code>ARRCOND=NO</code></td>
</tr>
<tr>
<td><code>ARRCOND=YES</code></td>
<td>Valid only when <code>ARR</code> is also coded.</td>
</tr>
<tr>
<td><code>ASYNCH=YES</code></td>
<td>Default: <code>ASYNCH=YES</code></td>
</tr>
<tr>
<td><code>ASYNCH=NO</code></td>
<td>Valid only when <code>ARR</code> is also coded.</td>
</tr>
<tr>
<td><code>CANCEL=YES</code></td>
<td>Default: <code>CANCEL=YES</code></td>
</tr>
<tr>
<td><code>CANCEL=NO</code></td>
<td>Valid only when <code>ARR</code> is also coded.</td>
</tr>
<tr>
<td><code>ASCMODE=PRIMARY</code></td>
<td>Default: <code>ASCMODE=PRIMARY</code></td>
</tr>
<tr>
<td><code>ASCMODE=AR</code></td>
<td></td>
</tr>
<tr>
<td><code>EAX=eax-value</code></td>
<td><code>eax-value</code>: Half-word decimal digit.</td>
</tr>
<tr>
<td><code>EK=entry-key</code></td>
<td><code>entry-key</code>: Decimal digit 0 - 15.</td>
</tr>
<tr>
<td><code>EKM=key-list</code></td>
<td><code>key-list</code>: List of keys or key ranges where a key is a number 0 - 15.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td><code>EKM</code> is required with <code>PKM=REPLACE</code>.</td>
</tr>
<tr>
<td><code>PARAM1=user-parm1</code></td>
<td><code>user-parm1</code>: A-type address or string of up to 4 characters enclosed by single quotation marks.</td>
</tr>
<tr>
<td><code>PARAM2=user-parm2</code></td>
<td><code>user-parm2</code>: A-type address or string of up to 4 characters enclosed by single quotation marks.</td>
</tr>
<tr>
<td><code>PC=STACKING</code></td>
<td>Default: <code>PC=STACKING</code></td>
</tr>
<tr>
<td><code>PC=BASIC</code></td>
<td></td>
</tr>
<tr>
<td><code>PROGRAM=pgm-name</code></td>
<td><code>pgm-name</code>: String of up to 8 alphanumeric characters, optionally enclosed by single quotation marks.</td>
</tr>
<tr>
<td><code>ROUTINE=rtn-addr</code></td>
<td><code>rtn-addr</code>: A-type address.</td>
</tr>
<tr>
<td><code>PKM=OR</code></td>
<td>Default: <code>PKM=OR</code></td>
</tr>
<tr>
<td><code>PKM=REPLACE</code></td>
<td></td>
</tr>
<tr>
<td><code>RAMODE=31</code></td>
<td>Default: <code>RAMODE=31</code></td>
</tr>
</tbody>
</table>
ETDEF macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,RAMODE=24</td>
<td></td>
</tr>
<tr>
<td>,RAMODE=64</td>
<td></td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>value: Any valid macro parameter specification.</td>
</tr>
<tr>
<td>,SASN=OLD</td>
<td>Default: SASN=OLD</td>
</tr>
<tr>
<td>,SASN=NEW</td>
<td></td>
</tr>
<tr>
<td>,SSWITCH=NO</td>
<td>Default: SSWITCH=NO</td>
</tr>
<tr>
<td>,SSWITCH=YES</td>
<td></td>
</tr>
<tr>
<td>,STATE=PROBLEM</td>
<td>Default: STATE=PROBLEM</td>
</tr>
<tr>
<td>,STATE=SUPERVISOR</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

The parameters are described as follows:

**TYPE=INITIAL**
Generates the header for the ETD.

**TYPE=ENTRY**
Generates an ETD entry. The system uses the defaults for any parameters you do not specify on the ETDEF TYPE=ENTRY macro. When you later specify ETDEF TYPE=SET, that macro initializes the entire ETD entry.

**TYPE=FINAL**
Specifies that the ETD is complete.

,**AKM=key-list**
Specifies a list of keys (0 through 15) or key ranges, optionally enclosed in parentheses, that identifies the authorized keys in which a problem program can use the PC routine. For example, AKM=(2,(3),5:8,(10:12),15) would authorize keys 2, 3, 5, 6, 7, 8, 10, 11, 12, and 15.

,**ARR=arr**
Specifies the associated recovery routine (ARR) that receives control if the stacking-PC routine abends. You can use the A-type address of the routine, or the name of the routine (an alphanumeric character string) enclosed in single quotation marks. If you use the name of the program, the program must be on the active LPA queue (FLPA or MLPA) or be in the PLPA or nucleus. The recovery routine will be entered in 31-bit mode. ARR is not valid with PC=BASIC.

,**ARRCOND=NO,** ARRCOND=YES
Specifies whether or not the ARR is conditional.

ARRCOND=NO, indicates that the ARR is not conditional, which means that the system follows the rules described in “Using ARRs” found in z/OS MVS Programming: Authorized Assembler Services Guide with respect to recording in LOGREC error recording if the ARR is skipped. ARRCOND=YES indicates that no recording in LOGREC error recording is to occur if the ARR is skipped.
Use ARRCOND=YES to avoid having to provide two PCs, one without an ARR for use in an FRR environment, and one with an ARR for use when not in an FRR environment.

ARRCOND is valid only with ARR.

,ASYNCH=YES
,ASYNCH=NO
Specifies whether or not the ARR can be interrupted by asynchronous exits. ASYNCH=YES specifies that the ARR can be interrupted by asynchronous exits. ASYNCH=NO specifies that the ARR cannot be interrupted by asynchronous exits. ASYNCH=YES is the default. ASYNCH is valid only with ARR.

,CANCEL=YES
,CANCEL=NO
Specifies whether or not the ARR can be interrupted by CANCEL/DETACH processing. CANCEL=YES specifies that the ARR can be interrupted by CANCEL/DETACH processing. CANCEL=NO specifies that the ARR cannot be interrupted by CANCEL/DETACH processing. CANCEL=YES is the default. CANCEL is valid only with ARR. To specify CANCEL=NO, one of the following conditions must be true for the stacking PC routine protected by the ARR:
- The stacking PC routine runs in supervisor state.
- The entry key for the stacking PC routine is a system key.
- The stacking PC routine runs with a system key valid for the entry key mask that will either replace or be ORed with the PKM.

,ASCMODE=PRIMARY
,ASCMODE=AR
Specifies that the stacking PC routine will execute in primary ASC mode (ASCMODE=PRIMARY) or in AR ASC mode (ASCMODE=AR). ASCMODE=AR is not valid with PC=BASIC. ASCMODE=PRIMARY is the default.

,EAX=eax-value
Specifies the extended authorization index (EAX) that the stacking PC routine uses. Specify an EAX that is owned by the home address space of the issuer of the ETCRE macro. An EAX of 'X'0000' means the PC routine is not EAX-authorized. If EAX is not specified, the PC routine has the same EAX as the issuer of the PC instruction. EAX is not valid with PC=BASIC.

,EK=entry-key
Specifies the PSW key (0 through 15) that the PC routine will run in. EK is not valid with PC=BASIC. If you omit EK, the PC routine gets control in the key of the caller.

,EKM=key-list
Specifies a list of keys (0 through 15) or key ranges, optionally enclosed in parentheses, that identify the entry key mask (EKM). When the PC routine is invoked, the keys specified identify either the additional keys that are to be ORed into the PKM (if PKM=OR is also specified or taken as the default) or the keys that should replace the PKM (if PKM=REPLACE is specified). EKM is required when you specify PKM=REPLACE.

,PARM1=user-parml
Specifies the address or character string to be placed in the first word of the latent parameter area associated with this ETD entry.
ETDEF macro

Addressability to the latent parameter area is through the current primary address space. The latent parameter address is set in general register 4 as a result of the PC instruction, although AR4 is unchanged by the PC instruction. If the PC routine runs in AR mode, set the access register corresponding to the latent parameter area to zero before the PC routine attempts to use it.

\text{\texttt{\textbackslash PARM2=\textbackslash user\textbackslash parm2}}

Specifies the address or character string to be placed in the second word of the latent parameter area associated with this ETD entry.

Addressability to the latent parameter area is through the current primary address space. The latent parameter address is set in general register 4 as a result of the PC instruction, although AR4 is unchanged by the PC instruction. If the PC routine runs in AR mode, set the access register corresponding to the latent parameter area to zero before the PC routine attempts to use it.

\text{\texttt{\textbackslash PC=\textbackslash STACKING}}

\text{\texttt{\textbackslash PC=\textbackslash BASIC}}

Indicates that this is a stacking PC (STACKING) or not a stacking PC (BASIC). Some parameters apply only to a stacking PC. STACKING is the default.

\text{\texttt{\textbackslash PROGRAM=\textbackslash pgm\textbackslash name}}

\text{\texttt{\textbackslash ROUTINE=\textbackslash rtn\textbackslash addr}}

Specifies the PC routine. When you specify PROGRAM, the PC routine must be on the active LPA queue (FLPA or MLPA) or be in the PLPA or nucleus. The same restriction applies also to ROUTINE, unless this is a space-switching PC or the PC is to be used only in the address space that established it. In other words, the PC routine for a space-switching PC can reside in the private area of the address space in which it will run, but the ROUTINE parameter must be used to specify it.

When you specify ROUTINE, you can indicate the AMODE of the PC routine with the RAMODE parameter. When you specify PROGRAM, the system locates the PC routine and determines its AMODE.

On TYPE=ENTRY or TYPE=SET,ETEADR, either PROGRAM or ROUTINE is required.

\text{\texttt{\textbackslash PKM=\textbackslash OR}}

\text{\texttt{\textbackslash PKM=\textbackslash REPLACE}}

Indicates either that the entry key mask (EKM) is ORed with the PSW key mask (PKM) or replaces the current PKM. PKM=REPLACE is not valid with PC=BASIC. PKM=OR is the default.

\text{\texttt{\textbackslash RAMODE=\textbackslash 31}}

\text{\texttt{\textbackslash RAMODE=\textbackslash 24}}

\text{\texttt{\textbackslash RAMODE=\textbackslash 64}}

Specifies the AMODE of the routine specified on the ROUTINE parameter. RAMODE is valid only with ROUTINE. If you specify PROGRAM rather than ROUTINE, the system locates the routine and determines its AMODE. RAMODE=31 is the default.

\text{\texttt{\textbackslash RELATED=value}}

Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user and may be any valid coding values.

\text{\texttt{\textbackslash SASN=\textbackslash OLD}}

\text{\texttt{\textbackslash SASN=\textbackslash NEW}}

Specifies whether the stacking PC routine will execute with SASN equal to the
ETDEF macro

caller's PASN (SASN=OLD), or with SASN equal to the PASN of the stacking
PC routine (SASN=NEW). SASN=NEW is not valid with PC=BASIC.
SASN=OLD is the default.

SSWITCH

SSWITCH=NO
SSWITCH=YES
Specifies whether or not the PC routine switches address spaces. If
SSWITCH=NO is specified, the PC does not switch address spaces. If
SSWITCH=YES is specified, the PC routine will execute in the address space of
the creator of the entry table with the authority of that address space.
SSWITCH=NO is the default.

STATE

STATE=PROBLEM
STATE=SUPERVISOR
Specifies which state the PC routine will receive control in either problem state
(PROBLEM) or supervisor state (SUPERVISOR). The default is
STATE=PROBLEM.

An example of using the ETDEF macro follows the description of the TYPE=SET
parameter.

TYPE=SET parameter

The ETDEF macro with the SET parameter works similarly to the execute form of a
macro with this important distinction: The TYPE=SET form totally replaces all
variables in an ETD entry and takes default values for all parameters you omit.
The normal execute form of a macro changes only the values you specify.

Constants and reserved fields that are initialized by other TYPE= forms are not
updated or changed. To create an entry table in a storage area that is not initialized
(for example, one just allocated through a GETMAIN request), you must first move
a complete entry table of the proper (or larger) size to that area. The formatted
table will provide the constants and indexes. Then, you can use ETDEF TYPE=SET
to change the required entry’s variable parameters.

Syntax

The form of SET is described as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ETDEF.</td>
</tr>
<tr>
<td>ETDEF</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ETDEF.</td>
</tr>
<tr>
<td>TYPE=SET,ETEADR=entry-addr</td>
<td>Required Parameters: PROGRAM or ROUTINE, AKM Valid Parameters: EKM, ARR, ASCMODE, EAX, EK, PARM1, PARM2, PC, PKM, RAMODE, SASN, SSWITCH, STATE, RELATED, ASYNCH, CANCEL</td>
</tr>
<tr>
<td>entry-addr</td>
<td>RX-type address or register (1) - (15).</td>
</tr>
</tbody>
</table>
## ETDEF macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>
| TYPE=SET,HEADER=header-addr | Required Parameter: NUMETE  
Valid Parameter: RELATED  
header-addr: RX-type address or register (1) - (15). |
| ,AKM=key-list | key-list: List of keys or key ranges where a key is a decimal digit 0 - 15. |
| ,ARR=arr | arr: A-type address, register (2)-(12), or alphanumeric character string, enclosed by single quotation marks. |
| ,ARRCOND=NO | Default: ARRCOND=NO |
| ,ARRCOND=YES | Valid only when ARR is also coded. |
| ,ASYNCH=YES | Default: ASYNCH=YES |
| ,ASYNCH=NO | Valid only when ARR is also coded. |
| ,CANCEL=YES | Default: CANCEL=YES  
Valid only when ARR is also coded. |
| ,CANCEL=NO | |
| ,ASCMODE=PRIMARY | Default: ASCMODE=PRIMARY  
,ASCMODE=AR |
| ,EAX=eax-value | eax-value: Half-word decimal digit or register (2)-(12) |
| ,EK=entry-key | entry-key: Decimal digit 0 - 15. |
| ,EKM=key-list | key-list: List of keys or key ranges where a key is a decimal digit 0 - 15.  
Note: EKM is required with PKM=REPLACE. |
| ,NUMETE=nbr-of-entries | nbr-of-entries: 2-byte A-type address, decimal number, or register (2)-(12).  
Note: NUMETE is required with HEADER. |
| ,PARM1=user-parm1 | user-parm1: A-type address, register (2)-(12), or string of up to 4 characters enclosed by single quotation marks. |
| ,PARM2=user-parm2 | user-parm2: A-type address, register (2)-(12), or string of up to 4 characters enclosed by single quotation marks. |
| ,PC=STACKING | Default: PC=STACKING  
,PC=BASIC |
ETDEF macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,PROGRAM=pgm-name</td>
<td>pgm-name: String of up to 8 alphanumeric characters, optionally enclosed by single quotation marks.</td>
</tr>
<tr>
<td>,ROUTINE=rtn-addr</td>
<td>rtn-addr: A-type address or registers (2)-(12)</td>
</tr>
<tr>
<td>,PKM=OR</td>
<td>Default: PKM=OR</td>
</tr>
<tr>
<td>,PKM=REPLACE</td>
<td></td>
</tr>
<tr>
<td>,RAMODE=31</td>
<td>Default: RAMODE=31</td>
</tr>
<tr>
<td>,RAMODE=24</td>
<td></td>
</tr>
<tr>
<td>,RAMODE=64</td>
<td></td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>value: Any valid macro parameter specification.</td>
</tr>
<tr>
<td>,SASN=OLD</td>
<td>Default: SASN=OLD</td>
</tr>
<tr>
<td>,SASN=NEW</td>
<td></td>
</tr>
<tr>
<td>,SSWITCH=NO</td>
<td>Default: SSWITCH=NO</td>
</tr>
<tr>
<td>,SSWITCH=YES</td>
<td></td>
</tr>
<tr>
<td>,STATE=PROBLEM</td>
<td>Default: STATE=PROBLEM</td>
</tr>
<tr>
<td>,STATE=SUPERVISOR</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

The parameters are described under the TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL options, with the following exceptions:

,ARRCOND=NO, ARRCOND=YES
Specifies whether or not the ARR is conditional.

ARRCOND=NO, which is the default, indicates that the ARR is not conditional, which means that if the system skips the ARR because of an incorrect environment, that fact is recorded in LOGREC error recording.
ARRCOND=YES indicates that if the system skips this ARR, that fact will not be recorded in LOGREC error recording. Use ARRCOND=YES to avoid having to provide two PCs, one without an ARR for use in an FRR environment, and one with an ARR for use when not in an FRR environment.

ARRCOND is valid only with ARR.

,NUMETE=nbr-of-entries
Specifies the number of contiguous entries in the ETD. nbr-of-entries is a decimal value from 1 to 256. NUMETE is required with the HEADER parameter. Use it to specify the number of entries you will use. It does not change the physical size of the table, but can be less than the initial size.

TYPE=SET, ETEADR=entry-addr
Specifies the address of the ETD entry. ETDEF TYPE=SET, ETEADR sets all the variable fields in the ETD entry that you generated through ETDEF.
ETDEF macro

TYPE=ENTRY macro. ETDEF TYPE=SET,ETEADR will set the ETD entry to the parameters you specify and to the defaults on all parameters you omit. That is, the system uses the default value, not the existing value, for any parameter that you omit.

**TYPE=SET, HEADER=header-addr**

Changes the size of the ETD. Use TYPE=SET, HEADER to decrease the size of the ETD from the size you originally established on ETDEF TYPE=INITIAL.

**ABEND codes**

None.

**Return and reason codes**

None.

**Example**

Define an entry table that has three entries. The PC routine called PCPGM receives control from a program with PSW key authorization of 8, the PC routine named OTHERTN receives control from a program with PSW authorization keys of 0 through 15, and the third PC routine called PCRTN receives control in PSW authorization key 0. The fourth ETDEF is there to show that the number of entries can be changed with ETDEF SET. (Perhaps, because of some input parameter, only a subset of all possible PC routines are set up. On another invocation of the program, perhaps all entries would be used.) The entries use all defaults other than those on the AKM parameter.

```assembly
MYPGM CSECT
   BALR 12, 0
   USING *, 12
   LOAD E=PCPGM
   LR 2, 0
   ETDEF TYPE=SET, HEADER=MYETDS, NUMETE=3
   ETDEF TYPE=SET, ETEADR=FIRST, ROUTINE=(2), AKM=8
   ETCRE ENTRIES=MYETDS
   RETURN
   .
   .
   .
   *
   DATA DEFINITIONS FOR PROGRAM
   .
MYETDS ETDEF TYPE=INITIAL
FIRST ETDEF TYPE=ENTRY, ROUTINE=0, AKM=8
SECOND ETDEF TYPE=ENTRY, PROGRAM=OTHERTN, AKM=0:15
THIRD ETDEF TYPE=ENTRY, ROUTINE=PCRTN, AKM=0
FOURTH ETDEF TYPE=ENTRY, ROUTINE=0, AKM=0
   ETDEF TYPE=FINAL
   *
   *
PCRTN DS OH
   .
   *
   PC ROUTINE CODE
   .
   .
END MYPGM
```

Note that the combination of TYPE=INITIAL, ENTRY, and FINAL is essentially the list form of the macro and TYPE=SET is the execute form.
Chapter 10. ETDES — Destroy entry table

Description

The ETDES macro is used to destroy a previously-created entry table.

Related macros

ETDEF, ETCRE, ETCON, and ETDIS

Environment

These are the requirements for the caller:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PKM 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN or PASN¬=HASN¬=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in primary address space</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

An entry table can be destroyed only by the address space that owns it.

Input register information

The ETDES macro is sensitive to the SYSSTATE macro with the OSREL=ZOSV1R6 parameter

- If the caller has issued the SYSSTATE macro with the OSREL=ZOSV1R6 parameter (Version 1 Release 6 of z/OS or later) before issuing the ETDES macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.
- Otherwise, the caller must ensure that the following general purpose register contains the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of an 18-word save area</td>
</tr>
</tbody>
</table>

Output register information

After the caller issues the macro, the macro might use some registers as work registers or might change the contents of some registers. When the macro returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers
ETDES macro

containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the macro</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the macro</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

Performance implications

None.

Syntax

The ETDES macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ETDES.</td>
</tr>
<tr>
<td>ETDES</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ETDES.</td>
</tr>
<tr>
<td>TOKEN=addr</td>
<td>addr: RX-type address or register (0) - (12).</td>
</tr>
<tr>
<td>,PURGE=NO</td>
<td>Default: PURGE=NO</td>
</tr>
<tr>
<td>,PURGE=YES</td>
<td></td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>value: Any valid macro keyword specification.</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

TOKEN=addr

Specifies the address of the fullword token (returned by the ETCRE macro) associated with the entry table to be destroyed.

,PURGE=NO
,PURGE=YES

Specifies whether (YES) or not (NO) the entry table is to be disconnected from all linkage tables and then destroyed.
ETDES macro

At the time ETDES is issued, the entry table must not be connected to any linkage tables unless PURGE=YES is coded. If any outstanding connections still exist and PURGE=YES is not coded, the entry table is not destroyed and the caller is abnormally terminated.

\texttt{\textbackslash RELATED=value}

Specifies information used to self-document macros by “relating” functions or services to corresponding services. The format and contents of the information specified can be any valid coding values.

### ABEND codes

052
053

See \textit{z/OS MVS System Codes} for an explanation and programmer responses for these codes.

### Return codes

When ETDES macro returns control to your program, GPR 15 contains a return code.

\textbf{Table 16. Return Codes for the ETDES Macro}

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: The specified entry table was destroyed. There were no connections to linkage indexes. Action: None required.</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: The specified entry table was destroyed. There were connections to linkage indexes, PURGE=YES was specified, and the entry table was disconnected. Action: None required. However, you may take some action based upon your application.</td>
</tr>
</tbody>
</table>

### Examples

For examples of the use of this and other cross memory macros, refer to the chapter on cross memory communication in \textit{z/OS MVS Programming: Extended Addressability Guide}.

### ETDES - List form

The list form of the ETDES macro constructs a nonexecutable parameter list. The execute form of the macro can refer to this parameter list, or a copy of it for reentrant programs.

### Syntax

The list form of the ETDES macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{name}</td>
<td>\texttt{name}: Symbol. Begin \texttt{name} in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ETDES.</td>
</tr>
</tbody>
</table>
ETDES macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETDES</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>One or more blanks must follow ETDES.</td>
</tr>
<tr>
<td>TOKEN=$addr$</td>
<td>$addr$: A-type address.</td>
</tr>
<tr>
<td>,PURGE=NO</td>
<td>Default: PURGE=NO</td>
</tr>
<tr>
<td>,PURGE=YES</td>
<td></td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>$value$: Any valid macro keyword specification.</td>
</tr>
<tr>
<td>,MF=L</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained under the standard form of the ETDES macro with the following exception:

,MF=L

Specifies the list form of the ETDES macro.

ETDES - Execute form

The execute form of the ETDES macro can refer to and modify a remote parameter list created by the list form of the macro.

Syntax

The execute form of the ETDES macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>$b$</td>
<td>One or more blanks must precede ETDES.</td>
</tr>
<tr>
<td>ETDES</td>
<td></td>
</tr>
<tr>
<td>$b$</td>
<td>One or more blanks must follow ETDES.</td>
</tr>
<tr>
<td>TOKEN=$addr$</td>
<td>$addr$: RX-type address or register (0) - (12).</td>
</tr>
<tr>
<td>,PURGE=NO</td>
<td>Default: PURGE=NO</td>
</tr>
<tr>
<td>,PURGE=YES</td>
<td></td>
</tr>
</tbody>
</table>
## ETDES macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,RELATED=value</td>
<td><em>value:</em> Any valid macro keyword specification.</td>
</tr>
<tr>
<td>,MF=(E,cntl addr)</td>
<td><em>cntl addr:</em> RX-type address or register (0) - (12).</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the ETDES macro with the following exception:

,\textbf{MF=} (\textbf{E},cntl addr)

Specifies the execute form of the ETDES macro. This form uses a remote parameter list.
ETDES macro
Chapter 11. ETDIS — Disconnect entry table

Description
The ETDIS macro disconnects one or more entry tables from the home address space's linkage table.

Related macros
ETDEF, ETCRE, ETCON, and ETDES

Environment
These are the requirements for the caller:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PKM 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN or PASN&lt;-&gt;HASN&lt;-&gt;SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in primary address space</td>
</tr>
</tbody>
</table>

Programming requirements
None.

Restrictions
An entry table, to be disconnected, must be connected to the home address space of the ETDIS issuer.

Input register information
The ETDIS macro is sensitive to the SYSSTATE macro with the OSREL=ZOSV1R6 parameter

- If the caller has issued the SYSSTATE macro with the OSREL=ZOSV1R6 parameter (Version 1 Release 6 of z/OS or later) before issuing the ETDIS macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.
- Otherwise, the caller must ensure that the following general purpose register contains the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of an 18-word save area</td>
</tr>
</tbody>
</table>

Output register information
After the caller issues the macro, the macro might use some registers as work registers or might change the contents of some registers. When the macro returns control to the caller, the contents of these registers are not the same as they were...
ETDIS macro

before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the macro</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the macro</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

Performance implications

None.

Syntax

The ETDIS macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ETDIS.</td>
</tr>
<tr>
<td>ETDIS</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ETDIS.</td>
</tr>
<tr>
<td>TKLIST=addr</td>
<td>addr: RX-type address or register (0) - (12).</td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>value: Any valid macro keyword specification.</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

**TKLIST=addr**

Specifies the address of a list of 1 to 32 fullword tokens, returned by the ETCRE macro, identifying the entry tables to be disconnected from the home address space’s linkage table. The first entry of the list must be a fullword count of the number of tokens (1 to 32) in the list.

,**RELATED=value**

Specifies information used to self-document macros by “relating” functions or services to corresponding services performed elsewhere. The format and contents of the information specified can be any valid coding values.
**ETDIS macro**

**ABEND codes**

052
053

See [z/OS MVS System Codes](https://www.ibm.com/support/knowledgecenter/en/A2FV41_2.3.0/com.ibm.zos.v2r3.sdk.cic.r04/pdf/zoscmpa0091052503.htm) for an explanation and programmer responses for these codes.

**Return codes**

When ETDIS macro returns control to your program, GPR 15 contains a return code.

*Table 17. Return Code for the ETDIS Macro*

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td><strong>Meaning</strong>: The entry table is successfully disconnected.</td>
</tr>
</tbody>
</table>

**Examples**

For examples of the use of this and other cross memory macros, refer to the chapter on cross memory communication in [z/OS MVS Programming: Extended Addressability Guide](https://www.ibm.com/support/knowledgecenter/en/A2FV41_2.3.0/com.ibm.zos.v2r3.sdk.cic.r04/pdf/zoscmpa0091052503.htm).
Chapter 12. EVENTS — Wait for one or more events to complete

Description

The EVENTS macro is a functional specialization of the WAIT macro with the ECBLIST parameter, with the advantages of notifying the program that events have completed and the order in which they completed.

The macro performs the following functions:

- Creates and deletes EVENTS tables.
- Initializes and maintains a list of completed event control blocks.
- Provides for single or multiple ECB processing.

The description of the EVENTS macro follows. The EVENTS macro is also described in [z/OS MVS Programming: Assembler Services Reference ABE-HSP](#) with the exception of the BRANCH=YES parameter.

Note: LOCAL lock means the local lock of the home address space.

Environment

The requirements for the caller are different for BRANCH=NO and BRANCH=YES.

If you specify BRANCH=NO, the requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state, with any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space</td>
</tr>
</tbody>
</table>

If you specify BRANCH=YES, the requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state and key 0</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>Local lock must be held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space</td>
</tr>
</tbody>
</table>

Programming requirements

If you specify BRANCH=YES, you must include the CVT mapping macro.
EVENTS macro

Restrictions
None.

Input register information
Before issuing the EVENTS macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications
None.

Syntax
The EVENTS macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede EVENTS.</td>
</tr>
<tr>
<td>EVENTS</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow EVENTS.</td>
</tr>
<tr>
<td>ENTRIES=n</td>
<td>n: Decimal digits 1-32767</td>
</tr>
<tr>
<td>ENTRIES=addr</td>
<td>addr: Register (2) - (12).</td>
</tr>
<tr>
<td>ENTRIES=DEL,TABLE=tab addr</td>
<td>tab addr: Symbol, RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>TABLE=tab addr</td>
<td>Note: If the ENTRIES parameter is specified as indicated in the first two formats, no other parameters may be specified.</td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,ECB=ecb addr</td>
<td><code>ecb addr</code>: Symbol, RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,LAST=last addr</td>
<td><code>last addr</code>: Symbol, RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: If LAST is specified, WAIT must also be specified.</td>
</tr>
<tr>
<td>,WAIT=YES</td>
<td><strong>Note</strong>: Do not specify WAIT=YES when running in a disabled state.</td>
</tr>
<tr>
<td>,WAIT=NO</td>
<td></td>
</tr>
<tr>
<td>,BRANCH=NO</td>
<td><strong>Default</strong>: BRANCH=NO</td>
</tr>
<tr>
<td>,BRANCH=YES</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained below:

- **ENTRIES=$n**
- **ENTRIES=addr**
  
  Specifies either a register or a decimal number from 1 to 32,767 which specifies the maximum number of completed ECB addresses that can be processed in an EVENTS table concurrently.

  **Note**: When this parameter is specified, no other parameter should be specified.

- **ENTRIES=DEL, TABLE=tab addr**
  
  Specifies that the EVENTS table whose address is specified by TABLE=tab addr is to be deleted. The user is responsible for deleting all of the tables he creates; however, all existing tables are automatically freed at task termination.

  **Note**:  
  1. When this parameter is specified, no other parameter should be specified.  
  2. TABLE resides in 24-bit addressable storage.

- **TABLE=tab addr**
  
  Specifies either a register number or the address of a word containing the address of the EVENTS table associated with the request. The address specified with the operand TABLE must be that of an EVENTS table created by this task.

  **Note**: TABLE resides in 24-bit addressable storage.

- **,WAIT=NO**
- **,WAIT=YES**
  
  Specifies whether or not to put the issuing program in a wait state when there are no completed events in the EVENTS table (specified by the TABLE parameter).

- **,ECB=ecb addr**
  
  Specifies either a register number or the address of a word containing the address of an event control block. The EVENTS macro should be used to initialize any event-type ECB. To avoid the accidental destruction of bit settings by a system service such as an access method, the ECB should be initialized.
EVENTS macro

after the system service that will post the ECB has been initiated (thus making
the ECB eligible for posting) and before the EVENTS macro is issued to wait
on the EVENTS table.

Note:
1. Register 1 should not be specified for the ECB address.
2. This parameter may not be specified with the LAST parameter.
3. The ECB can reside above or below 16 megabytes.
4. If only ECB initialization is being requested, neither WAIT=NO nor
   WAIT=YES should be specified, to prevent any unnecessary WAIT
   processing from occurring.

,.LAST=last addr
Specifies either a register number or the address of a word containing the
address of the last EVENT parameter list entry processed.

Note:
1. Do not specify Register 1 for the LAST address.
2. Do not specify this parameter with the ECB parameter.
3. The WAIT macro must also be specified.
4. LAST resides in 24-bit addressable storage.

.,BRANCH=NO
.,BRANCH=YES
Specifies that an SVC entry (BRANCH=NO) or a branch entry (BRANCH=YES)
is to be performed.

ABEND codes
The caller might encounter one of the following ABEND codes:
- 17A
- 17D
- 37A
- 37D
- 47A
- 47D
- 57D
- 67D
- 77D
- 87D

See z/OS MVS System Codes for explanations and responses for these codes.

Return and reason codes
None.

Example 1
The following shows total processing through EVENTS.

EVENTS and ECB Initialization
EVENTS macro

EVENTS ENTRIES=1000
ST R1,TABADD
WRITE ECBA
LA R2,ECBA...
EVENTS TABLE=TABADD,ECB=(R2)

Parameter List Processing

EVENTS TABLE=TABADD,WAIT=YES
LR R3,R1 PARMLIST ADDR
B LOOP2 GO TO PROCESS ECB
LOOP1 EVENTS TABLE=TABADD,WAIT=YES,LAST=(R3)
LR R3,R1 SAVE POINTER
LOOP2 EQU * PROCESS COMPLETED EVENTS
TM 0(R3),X'80' TEST FOR MORE EVENTS
BO LOOP1 IF NONE, GO WAIT
LA R3,4(,R3) GET NEXT ENTRY
B LOOP2 GO PROCESS NEXT ENTRY

Deleting EVENTS Table

EVENTS TABLE=TABADD,ENTRIES=DEL

TABADD DS F

Example 2
Processing One ECB at a Time.

EVENTS ENTRIES=10
ST 1,TABLE

NEXTREC GET TPDATA,KEY
ENQ (RESOURCE,ELEMENT,E,,SYSTEM)
READ DECBRW,KU,,'S',MF=E
LA 3,DECBRW
EVENTS TABLE=TABLE,ECB=(3),WAIT=YES
WRITE DECBRW,K,MF=E
LA 3,DECBRW
RETEST EVENTS TABLE=TABLE,ECB=(3),WAIT=NO
LTR 1,1
BNZ NEXTREC
B RETEST

TABLE DS F
EVENTS macro
Chapter 13. EXTRACT — Extract TCB information

Description

The EXTRACT macro causes the system to provide information from specified fields of the task control block or a subsidiary control block for either the active task or one of its subtasks. The system places the information in an area that the program provides. For a description of this area see “Providing an EXTRACT Answer Area” in z/OS MVS Programming: Authorized Assembler Services Guide.

When EXTRACT is issued, its parameter list can reside in 24 or 31-bit addressable storage.

To obtain the address of a TIOT entry, you can use either the GETDSAB macro or the EXTRACT macro.

Your installation might have installed products that require the use of the GETDSAB macro to obtain the address of the products’ TIOT entries. If you plan to use the EXTRACT macro, first check the documentation for the related product to ensure that the product does not require the use of the GETDSAB macro.

Note:

1. For procedures for using GETDSAB to obtain the address of a TIOT entry and the UCB address, see z/OS MVS Programming: Authorized Assembler Services Guide.

2. If the EXTRACT macro is used to obtain the TIOT in order to find the UCB, it is the user’s responsibility to ensure that the TIOT contains the UCB address. For procedures for finding the UCB address, see z/OS MVS Programming: Authorized Assembler Services Guide.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state, and user key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

None.

Performance implications

None.
EXTRACT macro

Syntax

The standard form of the EXTRACT macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: Symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede EXTRACT.</td>
</tr>
<tr>
<td>EXTRACT</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow EXTRACT.</td>
</tr>
<tr>
<td>answer addr</td>
<td><em>answer addr</em>: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,S'</td>
<td>Default: 'S'</td>
</tr>
<tr>
<td>,tcb addr</td>
<td><em>tcb addr</em>: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,FIELDS=(tcb info)</td>
<td><em>tcb info</em>: Any combination of the following, separated by commas: ALL, PRI, GRS, CMC, FRS, TIOT, AETX, COMM, TSO, PSB, TJID, ASID</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

- *answer addr*
  Specifies the address of the answer area to contain the requested information. The area is one or more fullwords, starting on a fullword boundary. The number of fullwords must be the same as the number of fields specified in the FIELDS parameter, unless ALL is coded. If ALL is coded, seven fullwords are required.

- ,S'
- ,tcb addr
  Specifies the address of a fullword on a fullword boundary containing the address of a task control block for a subtask of the active task. If 'S' is coded or is the default, no address is specified and the active task is assumed.

- ,FIELDS=(tcb info)
  Specifies the task control block information requested:
EXTRACT macro

ALL Requests information from the GRS, FRS, reserved, AETX, PRI, CMC, and TIOT fields. (If ALL is specified, 7 words are required just for ALL.)

GRS Is the address of the save area used by the system to save the general purpose registers 0-15 when the task is not active.

FRS Is the address of the save area used by the system to save the floating point registers 0, 2, 4, and 6 when the task is not active.

AETX Is the address of the end-of-task exit routine specified in the ETXR parameter of the ATTACH (or ATTACHX) macro used to create the task.

PRI Is the current limit (third byte) and dispatching (fourth byte) priorities of the task. The two high-order bytes are set to zero.

CMC Is the task completion code. If the task is not complete, the field is set to zero.

TIOT Is the address of the task input/output table.

COMM Is the address of the command scheduler communications list. The list consists of a pointer to the communications event control block and a pointer to the command input buffer, and a token. (If a token exists, the high-order bit of the token field is set to one). The token is used only with internal START commands. See “Issuing an Internal START or REPLY Command” in z/OS MVS Programming: Authorized Assembler Services Guide.

TSO Is the address of a byte in which a high-order bit of 1 indicates a TSO/E address space initiated from the LOGON command (that is, in a foreground TSO/E session). A high-order bit of 0 indicates either background TSO/E or a non-TSO/E address space.

PSB Is the address of the TSO/E protected step control block and is returned:

- In a foreground TSO/E session (initiated through LOGON)
- In a background TSO/E session (initiated through the TSO/E TMP, IKJEFT01).
- In a TSO/E environment initialized outside of the TSO/E TMP (initiated through the IKJTSOEV service).

TJID Is the address space identifier (ASID) for a foreground TSO/E session (initiated through LOGON), or zero for either background TSO/E or a non-TSO/E address space.

ASID Is the address space identifier.

ABEND codes
The EXTRACT macro might abnormally terminate with one of the following abend codes: X'128', X'228', and X'328'. See z/OS MVS System Codes for explanations and programmer responses.

Return and reason codes
None.
EXTRACT macro

Example 1
Provide information from all the fields of the indicated TCB except ASID. WHERE is the label of the answer area, ADDRESS is the label of a fullword that contains the address of the subtask TCB for which information is to be extracted.

```
EXTRACT WHERE,ADDRESS,FIELDS=(ALL,TSO,COMM,PSB,TJID)
```

Example 2
Provide information from the current TCB, as above.

```
EXTRACT WHERE,'S',FIELDS=(ALL,TSO,COMM,PSB,TJID)
```

Example 3
Provide information from the command scheduler communications list. ANSWER is the label of the answer area and TCBADDR is the label of a fullword that contains the address of the subtask TCB from which information is to be extracted.

```
EXTRACT ANSWER,TCBADDR,FIELDS=(COMM)
```

EXTRACT - List form

The list form of the EXTRACT macro is used to construct a remote control program parameter list.

Syntax
The list form of the EXTRACT macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>( b )</td>
<td>One or more blanks must precede EXTRACT.</td>
</tr>
<tr>
<td>( \text{EXTRACT} )</td>
<td></td>
</tr>
<tr>
<td>( b )</td>
<td>One or more blanks must follow EXTRACT.</td>
</tr>
<tr>
<td>answer addr</td>
<td>answer addr: A-type address.</td>
</tr>
<tr>
<td>,( S )</td>
<td>Default: 'S'</td>
</tr>
<tr>
<td>,tcb addr</td>
<td>tcb addr: A-type address.</td>
</tr>
<tr>
<td>,FIELDS=(tcb info)</td>
<td>tcb info: any combination of the following, separated by commas: ALL PRI GRS CMC FRS TIOT AETX COMM TSO PSB TJID ASID</td>
</tr>
</tbody>
</table>
### Parameters

The parameters are explained under the standard form of the EXTRACT macro, with the following exception:

\[ ,MF=L \]

Specifies the list form of the EXTRACT macro.

### EXTRACT - Execute form

The execute form of the EXTRACT macro uses, and can modify, a remote control program parameter list. If the FIELDS parameter, restricted in use, is coded in the execute form, any TCB information specified in a previous FIELDS parameter is canceled and must be respecified if required for this execution of the macro.

### Syntax

The execute form of the EXTRACT macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede EXTRACT.</td>
</tr>
<tr>
<td>EXTRACT</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow EXTRACT.</td>
</tr>
<tr>
<td>answer addr</td>
<td>answer addr: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>’S’</td>
<td>tcb addr: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,tcb addr</td>
<td></td>
</tr>
<tr>
<td>,FIELDS=(tcb info)</td>
<td>tcb info: any combination of the following, separated by commas: ALL PRI GRS CMC FRS TIOT AETX COMM TSO PSB TJID ASID</td>
</tr>
</tbody>
</table>
EXTRACT macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNF=(E,ctrl addr)</td>
<td><code>ctrl addr: RX-type address, or register (1) or (2) - (12).</code></td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained under the standard form of the EXTRACT macro, with the following exception:

```
, MNF=(E,ctrl addr)
```

Specifies the execute form of the EXTRACT macro using a remote control program parameter list.
Chapter 14. FESTAE — Fast extended STAE

Description
The FESTAE macro allows an SVC to define and activate, or to deactivate and no longer define, an ESTAE-type recovery routine with minimal overhead and no locking requirements. The ESTAE-type recovery routine activated by FESTAE receives control in the same sequence and under the same conditions as it would if it were activated by the ESTAE macro. The FESTAE macro can be issued in cross memory mode as long as the currently addressable address space is the home address space. For more information, see z/OS MVS Programming: Authorized Assembler Services Guide. To delete a FESTAE recovery routine that was established by the FESTAE macro, use the FESTAE macro rather than macros such as ESTAE, ESTAEX, or STAE.

The FESTAE macro expansion has no external linkage.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state and PSW key 0</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>The caller may hold locks, but is not required to hold any.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space Except for the TCB, all input parameters to this macro can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.</td>
</tr>
</tbody>
</table>

Programming requirements
FESTAE users executing in 31-bit addressing mode must recompile using the FESTAE macro expansion so that the exit routine gets control in 31-bit addressing mode.

The caller must include the following mapping macros:
- IHAPSA
- IHARB
- IHASCB
- IKJTCB

Restrictions
- Only type 2, 3, or 4 SVC routines can use the FESTAE macro
- The FESTAE macro can be issued to create only one recovery routine within the scope of the SVC routine. The ESTAEX macro or the ESTAE macro with the BRANCH option must be used to create additional recovery routines.
FESTAE macro

Input register information
Before issuing the FESTAE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register. Register notation is required for the following FESTAE macro parameters: EXITADR, WRKREG, RBADDR, TCBADDR, and PARAM.

Output register information
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unchanged</td>
</tr>
<tr>
<td>1-14</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>- If you specify 0,WRKREG=work reg addr, the register you specify (1-14) is used as a work register by the system.</td>
</tr>
<tr>
<td></td>
<td>- If you specify EXITADR=exit addr, the register you specify (1-14) is used as a work register by the system.</td>
</tr>
<tr>
<td></td>
<td>- Registers not specified for either work reg addr or exit addr are unchanged.</td>
</tr>
</tbody>
</table>

15 Return code

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>1-14</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications
Specification of the TCBADDR keyword results in more efficient code.

Syntax
The FESTAE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EXITADR=exit addr  exit addr: Register (1) - (14).
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,WRKREG=\textit{work reg}</td>
<td>\textit{work reg addr}: Register (1) - (14).</td>
</tr>
<tr>
<td>\texttt{RBADDR=svrb addr}</td>
<td>\textit{svrb addr}: Register (1) - (14).</td>
</tr>
<tr>
<td>\texttt{TCBADDR=tcb addr}</td>
<td>\textit{tcb addr}: Register (1) - (14).</td>
</tr>
<tr>
<td>\texttt{PARAM=list addr}</td>
<td>\textit{list addr}: Register (1) - (14).</td>
</tr>
<tr>
<td>\texttt{XCTL=NO}</td>
<td>Default: XCTL=NO</td>
</tr>
<tr>
<td>\texttt{XCTL=YES}</td>
<td></td>
</tr>
<tr>
<td>\texttt{PURGE=NONE}</td>
<td>Default: PURGE=NONE</td>
</tr>
<tr>
<td>\texttt{PURGE=HALT}</td>
<td></td>
</tr>
<tr>
<td>\texttt{PURGE=QUIESCE}</td>
<td></td>
</tr>
<tr>
<td>\texttt{ASYNCH=YES}</td>
<td>Default: ASYNCH=YES</td>
</tr>
<tr>
<td>\texttt{ASYNCH=NO}</td>
<td></td>
</tr>
<tr>
<td>\texttt{TERM=NO}</td>
<td>Default: TERM=NO</td>
</tr>
<tr>
<td>\texttt{TERM=YES}</td>
<td></td>
</tr>
<tr>
<td>\texttt{RECORD=NO}</td>
<td>Default: RECORD=NO</td>
</tr>
<tr>
<td>\texttt{RECORD=YES}</td>
<td></td>
</tr>
<tr>
<td>\texttt{ERRET=label}</td>
<td>\textit{label}: Any valid assembler name.</td>
</tr>
<tr>
<td>\texttt{SDWALOC31=NO}</td>
<td>Default: SDWALOC31=NO</td>
</tr>
<tr>
<td>\texttt{SDWALOC31=YES}</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

\texttt{EXITADR=exit addr}

\texttt{0,WRKREG=work reg}

Specifies whether an ESTAE-type recovery routine is to be defined and activated, or deactivated and no longer defined. \texttt{EXITADR=exit addr} specifies the register that contains the address of an ESTAE-type recovery routine to be entered if the task issuing FESTAE ends abnormally.

If you specify \texttt{0,WRKREG=work reg}, the current ESTAE-type recovery routine is deactivated and no longer defined if it was defined by the FESTAE macro. An error occurs if the current ESTAE-type recovery routine was not created by FESTAE. You do not have to initialize the register you specify for \textit{work reg}; the system uses it as a work register.
**FESTAE macro**

\[ ,RBADDR=svrb\ addr\]

Specifies a register that contains the address of the current SVRB prefix.
RBADDR must be specified if EXITADR has also been specified.

\[ ,TCBADDR=tcb\ addr\]

Specifies the register containing the current TCB address.

\[ ,PARAM=list\ addr\]

Specifies the register containing the address of a user-defined parameter list that contains data to be used by the ESTAE routine. The routine receives this address when it is scheduled for execution. The use of this parameter list is optional, but the user should zero out any spurious data it might contain whether or not he intends to use it. If the user does not select the PARAM option, the routine receives instead the 24-byte parameter area in the SVRB. In this case, the user must locate this SVRB parameter area and initialize it with appropriate data.

\[ ,ERRET=label\]

Specifies a label within the CSECT issuing the FESTAE for which addressability has been established. The FESTAE macro branches to this label if it is returning a code other than zero. This option saves the user the instructions necessary to check the return code. If the user does not specify the ERRET option, control returns instead to the instruction immediately following the FESTAE macro. The return code is in register 15.

All the other FESTAE parameters have the same meaning as their ESTAE counterparts.

**ABEND codes**

None.

**Return codes**

When control is returned to the instruction following the FESTAE macro, GPR 15 contains one of the following return codes.

<table>
<thead>
<tr>
<th>Table 18. Return Codes for the FESTAE Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hexadecimal Return Code</strong></td>
</tr>
<tr>
<td>00</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>08</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>0C</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Example

In case of an abnormal termination, execute the ESTAE routine specified by register 2, allow asynchronous processing, do not allow special error processing, default to PURGE=NONE, and pass the parameter list pointed to by register 7 to the ESTAE routine.

FESTAE EXITADR=(REG2),RBADDR=(REG3),TCBADDR=(REG6), PARAM=(REG7),ASYNCH=YES,TERM=NO
FESTAE macro
Chapter 15. FRACHECK — Check user's authorization (for RACF Release 1.8.1 or earlier)

See z/OS Security Server RACROUTE Macro Reference for a description of this macro.
Chapter 16. FREEMAIN — Free virtual storage

Description

Use the FREEMAIN macro to free one or more areas of virtual storage. You can also use the FREEMAIN macro to free an entire virtual storage subpool if it is owned by the task under which your program is issuing the FREEMAIN. For more information on releasing a subpool, see the chapter about virtual storage management in z/OS MVS Programming: Assembler Services Guide.

You can also use the STORAGE macro to free storage, even if the storage was obtained using the GETMAIN macro. Compared to FREEMAIN, STORAGE provides an easier-to-use interface and has no restrictions or locking requirements. See the chapter about virtual storage management in z/OS MVS Programming: Authorized Assembler Services Guide for a comparison of FREEMAIN and STORAGE.

The FREEMAIN macro is also described in z/OS MVS Programming: Assembler Services Reference ABE-HSP with the exception of the BRANCH parameter.

The FREEMAIN macro provides two types of entry linkage: SVC entry and branch entry. If you do not specify the BRANCH parameter, the FREEMAIN service receives control through SVC entry. If you specify the BRANCH parameter, the FREEMAIN service receives control through branch entry.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>For subpools 0-127: problem state and PSW key 8-15. For subpools 131 and 132, one or more of the following:</td>
</tr>
<tr>
<td></td>
<td>• Supervisor state</td>
</tr>
<tr>
<td></td>
<td>• PSW key 0-7</td>
</tr>
<tr>
<td></td>
<td>• APF-authorization.</td>
</tr>
<tr>
<td></td>
<td>• PSW key mask (PKM) that allows the calling program to switch its PSW key to match the key of the storage to be released.</td>
</tr>
<tr>
<td></td>
<td>For other subpools, one or more of the following:</td>
</tr>
<tr>
<td></td>
<td>• Supervisor state</td>
</tr>
<tr>
<td></td>
<td>• PSW key 0-7</td>
</tr>
<tr>
<td></td>
<td>• APF-authorized.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>For SVC entry: task. For branch entry: task or SRB.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>For SVC entry: PASN=HASN=SASN.</td>
</tr>
<tr>
<td></td>
<td>For branch entry: any PASN, any HASN, any SASN.</td>
</tr>
</tbody>
</table>
**FREEMAIN macro**

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AMODE:</strong> For SVC entry: 24- or 31- or 64-bit. For branch entry: 24- or 31-bit. For RU, RC requests: The system treats all addresses and values as 31-bit. For all other requests: If the calling program is in 31-bit mode, the system treats all addresses and values, passed to the FREEMAIN macro, as 31-bit. Otherwise, the system treats addresses and values as 24-bit.</td>
<td></td>
</tr>
<tr>
<td><strong>ASC mode:</strong> For BRANCH=(YES,GLOBAL), primary or access register (AR). For all other requests, primary. Callers in AR mode must use BRANCH=(YES,GLOBAL) and can obtain only global (common) storage.</td>
<td></td>
</tr>
<tr>
<td><strong>Interrupt status:</strong> For BRANCH=(YES,GLOBAL), disabled for I/O and external interrupts. For all other requests, enabled for I/O and external interrupts.</td>
<td></td>
</tr>
<tr>
<td><strong>Locks:</strong> For SVC entry, no locks may be held. For BRANCH=YES, your program must hold the local lock for the currently addressable address space. For BRANCH=YES, when running in cross-memory mode, your program must hold the CML lock for the currently addressable address space. For BRANCH=(YES,GLOBAL), your program must be in an MVS-recognized state of disablement, which can be achieved by obtaining the CPU lock.</td>
<td></td>
</tr>
<tr>
<td><strong>Control parameters:</strong> For LC, LU, L, VC, VU, V, EC, EU, E requests: control parameters must be in the primary address space. For other requests: control parameters are in registers.</td>
<td></td>
</tr>
</tbody>
</table>

**Programming requirements**

Before issuing the FREEMAIN macro in AR mode, issue SYSSTATE ASCENV=AR.

**Restrictions**

- Parameters passed to the FREEMAIN macro must not reside within the area being freed. If this restriction is violated and the parameters are the last allocated areas on a virtual page, the whole page is freed and FREEMAIN ends abnormally with an X’0C4’ abend code.
- The current task ends abnormally if the specified virtual storage area does not start on a doubleword boundary or, for an unconditional request, if the specified area or subpool is not owned by the task identified as the owner of the storage.
- For SVC entry, the caller cannot have an EUT FRR established.

**Input register information for SVC entry**

Before issuing the FREEMAIN macro without the BRANCH parameter (SVC entry), the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

**Output register information for SVC entry**

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
</tr>
</tbody>
</table>
Input register information for BRANCH=YES
Before issuing the FREEMAIN macro with BRANCH=YES, the caller must ensure that the following GPRs contain the specified information:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
</tr>
<tr>
<td>2-13</td>
</tr>
<tr>
<td>14-15</td>
</tr>
</tbody>
</table>

Output register information for BRANCH=YES
For RC, RU, VRC, and VRU requests: when control returns to the caller, GPRs contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4-13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>

For all other requests: when control returns to the caller, GPRs contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
</tr>
<tr>
<td>2-13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>

When control returns to the caller, ARs contain:
FREEMAIN macro

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1     Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13    Unchanged.</td>
</tr>
<tr>
<td>14-15   Used as work registers by the system.</td>
</tr>
</tbody>
</table>

Input register information for BRANCH=(YES,GLOBAL)

Before issuing the FREEMAIN macro with BRANCH=(YES,GLOBAL), you are not required to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information for BRANCH=(YES,GLOBAL)

When control returns to the caller, the GPRs contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1     Used as work registers by the system.</td>
</tr>
<tr>
<td>2       Unchanged.</td>
</tr>
<tr>
<td>3-4     Used as work registers by the system.</td>
</tr>
<tr>
<td>5-13    Unchanged.</td>
</tr>
<tr>
<td>14      Used as a work register by the system.</td>
</tr>
<tr>
<td>15      For a conditional request, contains the return code. For an unconditional request, used as a work register by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1     Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13    Unchanged.</td>
</tr>
<tr>
<td>14-15   Used as work registers by the system.</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the service returns control.

Performance implications

None.

Syntax

The standard form of the FREEMAIN macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede FREEMAIN.</td>
</tr>
<tr>
<td>FREEMAIN</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow FREEMAIN.</td>
</tr>
</tbody>
</table>
### FREEMAIN macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC,LA=ADDR</td>
<td><em>length addr</em>: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>LU,LA=ADDR</td>
<td></td>
</tr>
<tr>
<td>L,LA=ADDR</td>
<td></td>
</tr>
<tr>
<td>VC</td>
<td></td>
</tr>
<tr>
<td>VU</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>EC,LV=VALUE</td>
<td><em>length value</em>: symbol, decimal number, or register (2) - (12).</td>
</tr>
<tr>
<td>EU,LV=VALUE</td>
<td></td>
</tr>
<tr>
<td>E,LV=VALUE</td>
<td></td>
</tr>
<tr>
<td>RC,LV=VALUE</td>
<td>If R, RC, or RU is specified, register (0) may also be used.</td>
</tr>
<tr>
<td>RC,SP=POOL</td>
<td><em>subpool nmbr</em>: symbol, decimal number 0-255, or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td>If R is specified, register (0) may also be used.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: For a subpool release (RC,SP or RU,SP, or R,SP), no other parameters except RELATED and BRANCH=YES can be specified.</td>
</tr>
<tr>
<td>RU,LV=VALUE</td>
<td></td>
</tr>
<tr>
<td>RU,SP=POOL</td>
<td></td>
</tr>
<tr>
<td>R,LV=VALUE</td>
<td></td>
</tr>
<tr>
<td>R,SP=POOL</td>
<td></td>
</tr>
<tr>
<td>,A=ADDR</td>
<td><em>addr</em>: A-type address, or register (2) - (12). If R, RC, or RU is specified, register (1) can also be used.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: If R, RC, or RU is specified, register (1) can also be specified.</td>
</tr>
<tr>
<td>,SP=POOL</td>
<td><em>subpool nmbr</em>: symbol, decimal number 0-255, or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td><strong>Default</strong>: SP=0. If R is specified, register (0) may also be used.</td>
</tr>
<tr>
<td>,BRANCH=YES</td>
<td><strong>Note</strong>: BRANCH=(YES,GLOBAL) may be specified only with RC or RU.</td>
</tr>
<tr>
<td>,BRANCH=(YES,GLOBAL)</td>
<td></td>
</tr>
<tr>
<td>,KEY=NUMBER</td>
<td><em>nmbr</em>: decimal numbers 0-15, or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: KEY may be specified only with RC or RU.</td>
</tr>
<tr>
<td>,RELATED=VALUE</td>
<td><em>value</em>: any valid assembler character string.</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

- **LC,LA=ADDR**
- **LU,LA=ADDR**
- **L,LA=ADDR**

---

Chapter 16. FREEMAIN — Free virtual storage  171
FREEMAIN macro

VC
VU
V
EC, LV= length value
EU, LV= length value
E, LV= length value
RC, LV= length value
RU, LV= length value
RU, SP= subpool nmbr
R, LV= length value
R, SP= subpool nmbr

Specifies the type of FREEMAIN request:

LC, LU, and L indicate conditional (LC) and unconditional (LU and L) list requests and specify release of one or more areas of virtual storage. The length of each virtual storage area is indicated by the values in a list beginning at the address specified in the LA parameter. The address of each of the virtual storage areas must be provided in a corresponding list whose address is specified in the A parameter. All virtual storage areas must start on a doubleword boundary.

VC, VU, and V indicate conditional (VC) and unconditional (VU and V) variable requests and specify release of single areas of virtual storage. The address and length of the virtual storage area are provided at the address specified in the A parameter.

EC, EU, and E indicate conditional (EC) and unconditional (EU and E) element requests and specify release of single areas of virtual storage. The length of the single virtual storage area is indicated in the LV parameter. The address of the virtual storage area is provided at the address indicated in the A parameter.

RC, RU, and R indicate conditional (RC) and unconditional (RU and R) register requests and specify either the release of all the storage in a subpool or the release of a certain area in a subpool. For information on how to release all the storage in a subpool, see the description for the SP parameter. If the release is for a certain area in a subpool, the address of the virtual storage area is indicated in the A parameter. The length of the area is indicated in the LV parameter. The virtual storage area must start on a doubleword boundary.

Note:

1. For a conditional request, errors detected while processing a FREEMAIN request with incorrect or inconsistent parameters cause the FREEMAIN service to return to the caller with a non-zero return code. For all other errors, the system abnormally ends the active task if the FREEMAIN request cannot be successfully completed.

   For an unconditional request, the system abnormally ends the active task if the FREEMAIN request cannot be successfully completed.

2. If the address of the area to be freed is above 16 megabytes, you must use RC or RU.

LA specifies the virtual storage address of one or more consecutive fullwords starting on a fullword boundary. One word is required for each virtual storage area to be released; the high-order bit in the last word must be set to 1 to indicate the end of the list. Each word must contain the required length in the low-order three bytes. The fullwords in this list must correspond with the fullwords in the associated list specified in the A parameter. The words must not be in the area to be released. If this rule is violated and if the words are the
last allocated items on a virtual page, the whole page is returned to storage
and the FREEMAIN abends with an X'0C4' abend code.

LV specifies the length, in bytes, of the virtual storage area being released. The
value should be a multiple of 8; if it is not, the control program uses the next
high multiple of 8.
• If you specify R,LV=(0) you cannot specify the SP parameter. You must
specify the subpool in register 0; the high-order byte must contain the
subpool number and the low-order three bytes must contain the length
unless you are requesting a subpool release. On a subpool release, the
low-order three bytes must contain zeros.
• If you specify R,LV using a symbol, decimal number, or register 2-12, you
can specify the SP parameter using registers 0 or 2-12.

\[A=\text{addr}\]
Specifies the virtual storage address of one or more consecutive fullwords
starting on a fullword boundary.
• If E, EC, or EU is coded, one word is required, which contains the address of
the virtual storage area to be released.
• If V, VC, or VU is coded, two words are required; the first word contains the
address of the virtual storage area to be released, and the second word
contains the length of the area to be released.
• If L, LC, or LU is coded, one word is required for each virtual storage area
to be released; each word contains the address of one virtual storage area.
• If R, RC, or RU is coded, one word is required, which contains the address
of the virtual storage area to be released. If R, RC, or RU is coded and \(\text{addr}\)
specifies a register, register 1 through 12 can be used and must contain the
address of the virtual storage area to be released.

Do not specify a storage address of 0 with a storage length of 0. This
combination causes FREEMAIN to free the subpool specified with the SP
parameter, or subpool 0 if the SP parameter is omitted.

\[SP=\text{subpool nmbr}\]
Specifies the subpool number of the virtual area to be released. Valid subpools
numbers are between 0 and 255. The SP parameter is optional and if omitted,
subpool 0 is assumed. If you specify a register, the subpool number must be in
bits 24-31 of the register, with bits 0-23 set to zero.

A request to release all the storage in a subpool is known as a subpool release.
To issue a subpool release, specify RC,SP or RU,SP or R,SP; and do not use the
A or the KEY parameter. The following subpools are valid on the SP parameter
for a subpool release: 0-127, 129-132, 203-204, 213-214, 223-224, 229-230, 233,
236-237, 240, 249, and 250-253. An attempt to issue a subpool release for any
other subpool causes an abend X'478' or X'40A'. For information about
subpools, see z/OS MVS Programming: Assembler Services Guide and z/OS MVS
Programming: Authorized Assembler Services Guide.

Note:
1. Callers executing in supervisor state and PSW key 0, who specify subpool
0, will free storage from subpool 252. Therefore, when requesting a dump
of this storage through the SDUMP macro, the caller must specify subpool
252 rather than subpool 0.
2. Requests for storage from subpools 240 and 250 are translated to subpool 0
storage requests.
FREEMAIN macro

\texttt{,BRANCH=(YES,GLOBAL)}
Specifies that a branch entry is to be used.

\texttt{BRANCH=YES} allows both local (private area) and global (common area) storage to be released. See “Input register information for \texttt{BRANCH=\textit{YES}}” on page 169 for specific information on input register requirements.

\texttt{BRANCH=(YES,GLOBAL)} allows only global storage to be released. With \texttt{BRANCH=(YES,GLOBAL)}, the SP parameter may designate only subpools 226-228, 231, 239, 241, 245, 247, or 248. \texttt{BRANCH=(YES,GLOBAL)} is valid only with RC or RU.

\texttt{,KEY=\textit{key number}}
Specifies the storage key in which the storage was obtained. The valid storage keys are 0-15. If a register is specified, the storage key must be in bits 24-27 of the register. \texttt{KEY} can be specified for the following subpools: 129-132, 227-231, 241, and 249. \texttt{BRANCH} is required with \texttt{KEY} for subpools 227-231, 241, and 249. \texttt{BRANCH=(YES,GLOBAL)} is not valid for subpools 129-132, 229-230, and 249.

\texttt{,RELATED=value}
Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services and can be any valid assembler character string.

\textbf{ABEND codes}
Abend codes FREEMAIN might issue are listed below in hexadecimal. For detailed abend code information, see \textit{z/OS MVS System Codes}

\begin{itemize}
  \item 105
  \item 10A
  \item 178
  \item 205
  \item 20A
  \item 278
  \item 305
  \item 30A
  \item 378
  \item 40A
  \item 478
  \item 505
  \item 605
  \item 705
  \item 70A
  \item 778
  \item 805
  \item 80A
  \item 878
  \item 905
  \item 90A
  \item 978
  \item A05
\end{itemize}
Return and reason codes

When the FREEMAIN macro returns control to your program and you specified a conditional request, GPR 15 contains one of the following hexadecimal return codes:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><strong>Meaning</strong>: Successful completion.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: None.</td>
</tr>
<tr>
<td>4</td>
<td><strong>Meaning</strong>: Program error. Not all requested virtual storage was freed.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check your program for the following kinds of errors:</td>
</tr>
<tr>
<td></td>
<td>• The address of the storage area to be freed is not correct.</td>
</tr>
<tr>
<td></td>
<td>• The subpool you have specified does not match the subpool of the storage to be freed.</td>
</tr>
<tr>
<td></td>
<td>• The key you have specified does not match the key of the storage to be freed.</td>
</tr>
<tr>
<td></td>
<td>• For private storage: the owning task identified by the input TCB is not correct for the storage to be freed.</td>
</tr>
<tr>
<td>8</td>
<td><strong>Meaning</strong>: Program error. No virtual storage was freed because part of the storage area to be freed is fixed.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Determine whether you have made one of the following errors. If so, correct your program and rerun it:</td>
</tr>
<tr>
<td></td>
<td>• You passed an incorrect storage area address to the FREEMAIN macro.</td>
</tr>
<tr>
<td></td>
<td>• You attempted to free storage that is fixed.</td>
</tr>
</tbody>
</table>

Example 1

Free 400 bytes of storage from subpool 10. Register 1 contains the address of the storage area. If the storage is not allocated to the current task, do not abnormally terminate the caller.

FREEMAIN RC, LV=400, A=(1), SP=10

Example 2

Free all of subpool 3 (if any) that belongs to the current task. If the request is not successful, abnormally terminate the caller.

FREEMAIN RU, SP=3

Example 3

Free from subpool 5, three areas of storage of 200, 800, and 32 bytes, previously obtained using the list and execute forms of the GETMAIN macro. Storage area addresses are in AREAADD. If any of the storage areas to be freed are not allocated to the current task, abnormally terminate the caller.
**FREEMAIN macro**

```assembly
FREE MAIN LU,LA=L NTH L IS T,A=A R EA ADD,SP=5
  .
  .
L NTH L IS T DC F'200',F'800',X'80',FL3'32'
AREA ADD DS 3F
```

**Example 4**
Free 400 bytes of storage from default subpool 0 using branch entry. The address of the storage area is in register 2. If the request is not successful, do not abnormally terminate the caller.

```assembly
FREE MAIN EC,LV=400,A=(2),BRANCH=YES
```

**Example 5**
Free 48 bytes of storage from subpool 231 using global branch entry. Register 5 contains the address of the storage area. Register 3 contains the storage key of the storage to be released. If the request is unsuccessful, abnormally terminate the caller.

```assembly
FREE MAIN RU,LV=48,A=(5),SP=231,KEY=(3),BRANCH=(YES,GLOBAL)
```

**FREEMAIN - List form**
Use the list form of the FREEMAIN macro to construct a nonexecutable control program parameter list.

The list form of the FREEMAIN macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede FREEMAIN.</td>
</tr>
<tr>
<td>FREEMAIN</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow FREEMAIN.</td>
</tr>
<tr>
<td>LC</td>
<td></td>
</tr>
<tr>
<td>LU</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td></td>
</tr>
<tr>
<td>VC</td>
<td></td>
</tr>
<tr>
<td>VU</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>EC</td>
<td></td>
</tr>
<tr>
<td>EU</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>

```assembly
,LA=length addr
```

`length addr`: A-type address.
FREEMAIN macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,LV=length value</td>
<td>length value: symbol or decimal number.</td>
</tr>
</tbody>
</table>

**Note:**
1. LA may only be specified with LC, LU, or L above.
2. LV may only be specified with EC, EU, or E above.

,LA=addr       addr: A-type address.

,SP=subpool nmbr  subpool nmbr: symbol or decimal number.

,RELATED=value   value: any valid assembler character string.

,MF=L

Parameters

The parameters are explained under the standard form of the FREEMAIN macro, with the following exceptions:

,MF=L

Specifies the list form of the FREEMAIN macro.

FREEMAIN - Execute form

A remote control program parameter list is used in, and can be modified by, the execute form of the FREEMAIN macro. The parameter list can be generated by the list form of either a GETMAIN or a FREEMAIN.

The execute form of the FREEMAIN macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
</tbody>
</table>

b
One or more blanks must precede FREEMAIN.

FREEMAIN

b
One or more blanks must follow FREEMAIN.

LC
LU
L
VC
VU
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>EC</td>
<td></td>
</tr>
<tr>
<td>EU</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>,LA=length addr</td>
<td>length addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,LV=length value</td>
<td>length value: symbol, decimal number, or register (2) - (12).</td>
</tr>
</tbody>
</table>

**Note:**
1. LA may only be specified with LC, LU, or L above.
2. LV may only be specified with EC, EU, or E above.

| A=addr            | addr: RX-type address, or register (2) - (12).                            |
| SP=subpool nmbr   | subpool nmbr: symbol, decimal number, or register (0) or (2) - (12).     |
| ,BRANCH=YES       |                                                                             |
| ,RELATED=value    | value: any valid assembler character string.                              |
| ,MF=(E,list addr) | list addr: RX-type address, or register (1) or (2) - (12).               |

### Parameters

The parameters are explained under the standard form of the FREEMAIN macro, with the following exceptions:

**,MF=(E,list addr)**

Specifies the execute form of the FREEMAIN macro using a remote control program parameter list.
Chapter 17. GETDSAB — Accessing the DSAB chain

Description

The GETDSAB macro returns a pointer to the data set association block (DSAB) associated with a DD name.

Use the GETDSAB macro to:
- Retrieve the address of the first DSAB associated with a DD name, as specified by:
  - An input DD name
  - An input device control block (DCB) address
  - An input task control block (TCB) address.
- Scan the DSAB chain

See [z/OS MVS Programming: Authorized Assembler Services Guide](#) for procedures that use the DSAB address returned by the GETDSAB macro to obtain the address of the TIOT entry and the UCB address.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state or supervisor state, and any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit addressing mode</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space</td>
</tr>
</tbody>
</table>

Programming requirements

To use GETDSAB, the caller must include the DSAB mapping macro (IHADSAB), the CVT mapping macro (CVT), and the JESCT mapping macro (IEFJESCT).

The caller must provide or inherit serialization on the SYSZTIOT resource before calling the GETDSAB macro and while using the output addresses of the macro. The minimum required level of serialization is shared (SHR).

The GETDSAB service does not provide a recovery environment. Because the service runs in task mode, the system will use any recovery environment defined to the caller prior to invoking GETDSAB.

Restrictions

There are no restrictions or limitations for using this macro.
GETDSAB macro

Register information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

On input to the macro, register 13 must contain the address of an 18-word save area.

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR 15 contains 12; otherwise, used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

Performance implications

There are no performance implications related to GETDSAB.

Syntax

The standard form of the GETDSAB macro follows.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede GETDSAB.</td>
</tr>
<tr>
<td>GETDSAB</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow GETDSAB.</td>
</tr>
<tr>
<td>FIRST</td>
<td></td>
</tr>
<tr>
<td>NEXT</td>
<td></td>
</tr>
<tr>
<td>DCBPTR=dcb addr</td>
<td>dcb addr: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>DDNAME=dd addr</td>
<td>dd addr: RX-type address, or register (2) - (12). This address specifies an 8-byte field which contains a DD name.</td>
</tr>
<tr>
<td>DSABPTR=dsab addr</td>
<td>dsab addr: RX-type address, or register (2) - (12).</td>
</tr>
</tbody>
</table>
### Parameters

The parameters are described as follows:

- **FIRST**
- **NEXT**
- **DCBPTR=dcb addr**
- **DDNAME=dd addr**

**DCBPTR=dcb addr** specifies the name of a pointer that contains the address of a fullword field. The fullword points to the DCB associated with a DD name. The system retrieves the DSAB pointer associated with the DCB.

When DCBPTR points to an open DCB, DCBPTR and TCBPTR are mutually exclusive.

Do not use the DCBPTR option for a DCB in a DCB OPEN exit, DCB ABEND exit, data management ABEND installation exit or the DCB OPEN installation exit.

When DCBPTR points to a closed DCB, the system selects the DSAB chain associated with the TCB specified by TCBPTR parameter, or, if none is specified, by the current TCB.

**DDNAME=dd addr** specifies a DD name associated with a DSAB. The system puts the address of the DSAB associated with this DD name into the fullword field specified by the DSABPTR parameter. The DSAB selected is that associated with the TCB specified by the TCBPTR parameter, or, if none is specified, by the current TCB. The *dd addr* is an 8-character, left-justified field, with trailing blanks. The *dd addr* may not contain all blanks.

**DSABPTR=dsab addr**

Specifies the name of a required fullword field that will be set to the address of the desired DSAB.

When used with the NEXT keyword, DSABPTR must contain the address of a DSAB that was previously obtained by invoking GETDSAB with FIRST, DCBPTR, or DDNAME. The system will replace this initial address with the address of the next DSAB in the DSAB chain.

### Syntax Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCBPTR=tcb addr</td>
<td><em>tcb addr</em>: RX-type address, or register (2) - (12). <strong>Default</strong>: TCBPTR=0</td>
</tr>
<tr>
<td>RETCODE=retcode addr</td>
<td><em>retcode addr</em>: RX-type address, or register (2) - (12) of fullword output variable</td>
</tr>
<tr>
<td>RSNODE=rname addr</td>
<td><em>name</em>: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>LOC=BELOW</td>
<td><strong>Default</strong>: BELOW</td>
</tr>
<tr>
<td>LOC=ANY</td>
<td></td>
</tr>
</tbody>
</table>

---

**GETDSAB macro**

---

Chapter 17. GETDSAB — Accessing the DSAB chain 181
GETDSAB macro

When used with the keywords FIRST, DCBPTR, or DDNAME, DSABPTR is an output field only.

Upon output, DSABPTR contains the address of the specified DSAB if the return code is zero. If the return code is not zero, DSABPTR contains 0.

\texttt{,TCBPTR=tcb\ addr}

Specifies the name of a pointer that contains the address of the TCB associated with the task for which DSAB information is requested.

When DCBPTR points to an open DCB, DCBPTR and TCBPTR are mutually exclusive.

The default, TCBPTR=0, requests the current TCB.

\texttt{,RETCODE=retcode\ addr}

Specifies the location where the system is to store the return code. The return code is also in GPR 15.

\texttt{,RSNCODE=rsncode\ addr}

Specifies the location where the system is to store the reason code. The reason code is also in GPR 0.

\texttt{LOC=BELOW}

\texttt{LOC=ANY}

Specifies whether or not GETDSAB should limit where it searches to find the DSAB corresponding to the input DDname.

LOC=BELOW, which is the default, searches only those DSABs residing below the 16 MB line.

LOC=ANY searches both below and above the 16 MB line.

Return and reason codes

When control returns from GETDSAB, GPR 15 (and \texttt{retcode\ addr}, if you coded RETCODE) contains one of the following decimal return codes:

<table>
<thead>
<tr>
<th>Decimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: Successful completion</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: Request failed. NEXT was specified when DSABPTR pointed to the last DSAB in the DSAB chain.</td>
</tr>
<tr>
<td>08</td>
<td>Meaning: Request failed. The specified DSAB was not found.</td>
</tr>
<tr>
<td>12</td>
<td>Meaning: Request failed. Input values were in error or in conflict.</td>
</tr>
<tr>
<td>16</td>
<td>Meaning: Request failed. The GETDSAB function is not currently installed on the system. Consult your system programmer.</td>
</tr>
</tbody>
</table>

When control returns from GETDSAB, GPR 0 (and \texttt{rsncode\ addr}, if you coded RSNCODE) might contain one of the following decimal reason codes:

<table>
<thead>
<tr>
<th>Decimal Return Code</th>
<th>Decimal Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1200</td>
<td>Request failed because of input error. The DDNAME specified or obtained was all blanks.</td>
</tr>
<tr>
<td>12</td>
<td>1210</td>
<td>Request failed because of input error. TCBPTR was specified when DCBPTR points to an open DCB.</td>
</tr>
</tbody>
</table>
Table 21. Return and Reason Codes for the GETDSAB Macro (continued)

<table>
<thead>
<tr>
<th>Decimal Return Code</th>
<th>Decimal Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1220</td>
<td>Request failed because of input error. The DSAB pointed to by DSABPTR is not valid.</td>
</tr>
<tr>
<td>12</td>
<td>1230</td>
<td>Request failed because of input error. The LOC=BLOW is requested, but the DSAB chain contains DSABs from both above and below the 16 MB line. Use LOC=ANY for this DSAB chain.</td>
</tr>
</tbody>
</table>

Example 1

In this example, MYDSAB will contain the address of DSAB associated with the DD named DD09.

```
MVC THEDD,'CL8'='DD09'
GETDSAB DDNAME=THEDD,DSABPTR=MYDSAB
         ...
AUTO    DSECT
THEDD   DS  CL8
MYDSAB  DS  AL4
```

Example 2

In this example, the first invocation of GETDSAB will set MYDSAB to the address of the first DSAB in the DSAB chain. MYRC will contain the return code.

The second invocation of GETDSAB will replace the initial address in MYDSAB with the address of the next DSAB in the DSAB chain.

```
GETDSAB FIRST,DSABPTR=MYDSAB,RETCODE=MYRC
         ...
GETDSAB NEXT,DSABPTR=MYDSAB
         ...
AUTO    DSECT
MYDSAB  DS  AL4
MYRC    DS  F
```

This technique can be used to get the DSAB for the first DD in a concatenation and then to step through the DSABs for all other DDs in the concatenation. It is the user's responsibility to determine when the DSAB for the last DD in the concatenation has been fetched, because a subsequent invocation of GETDSAB NEXT will simply return the next DSAB on the chain (if one exists), even if it is for a different DD statement.

Example 3

In this example, DCBPTR contains the address of a fullword pointer that points to the DCB associated with a DD name. MYDSAB will contain the address of the DSAB associated with the DCB. MYRSN will contain the reason code.

```
GETDSAB DSBPTR=MYDCB,DSABPTR=MYDSAB,RSNCODE=MYRSN
         ...
```
Example 4

If DCBPTR points to an open DCB, DCBPTR and TCBPTR are mutually exclusive.
The request will fail with return code 12. MYDSAB will contain 0.

If DCBPTR points to a closed DCB, the system will search the DSAB chain
associated with the TCB. MYDSAB will contain the address of the DSAB related to
the TCB specified by TCBPTR.

GETDSAB DCBPTR=MYDCB,DSABPTR=MYDSAB,TCBPTR=MYTCB

GETDSAB - List form

Use the list form of the GETDSAB macro together with the execute form of the
macro for applications that require reentrant code. The list form of the macro
defines an area of storage, which the execute form of the macro uses to store the
parameters.

Syntax

The list form of the GETDSAB macro follows.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede GETDSAB.</td>
</tr>
<tr>
<td>GETDSAB</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow GETDSAB.</td>
</tr>
<tr>
<td>MF=(L,stor addr)</td>
<td>stor addr: symbol.</td>
</tr>
<tr>
<td>MF=(L,stor addr,attr)</td>
<td>attr: 1- to 60-character input string. Default: 0D</td>
</tr>
</tbody>
</table>

Parameters

The following parameters are the only ones you can specify using the list format:

MF=L

Specifies the list form of the GETDSAB macro.
The `stor addr` parameter specifies the name of a required storage area for the parameter list. This storage area will be generated as part of the macro expansion and should not be separately defined by the user. Note also, that the "stor addr" in the List and Execute forms of the macro must refer to the same storage area.

`attr` is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code `attr`, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

**GETDSAB - Execute form**

Use the execute form of the GETDSAB macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

**Syntax**

The execute form of the GETDSAB macro follows.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: Symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede GETDSAB.</td>
</tr>
<tr>
<td>GETDSAB</td>
<td></td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must follow GETDSAB.</td>
</tr>
<tr>
<td><strong>FIRST</strong></td>
<td></td>
</tr>
<tr>
<td><strong>NEXT</strong></td>
<td></td>
</tr>
<tr>
<td><code>DCBPTR=dcb addr</code></td>
<td><code>dcb addr</code>: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td><code>DDNAME=dd addr</code></td>
<td><code>dd addr</code>: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td><code>DSABPTR=dsab addr</code></td>
<td><code>dsab addr</code>: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td><code>TCBPTR=tcb addr</code></td>
<td><code>tcb addr</code>: RX-type address, or register (2) - (12). <strong>Default</strong>: TCBPTR=0</td>
</tr>
<tr>
<td><code>RETCODE=retcode addr</code></td>
<td><code>retcode addr</code>: RX-type address, or register (2) - (12) of fullword output variable</td>
</tr>
<tr>
<td><code>RSNCODE=rsncode addr</code></td>
<td><code>rsncode addr</code>: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td><code>LOC=BETWEEN</code></td>
<td><strong>Default</strong> = BETWEEN</td>
</tr>
<tr>
<td><code>LOC=ANY</code></td>
<td></td>
</tr>
</tbody>
</table>
## GETDSAB macro

### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF=(E,stor addr)</td>
<td>stor addr: RX-type address, or any register (1) - (12). If register 1 is specified, its value may be changed by the macro invocation.</td>
</tr>
<tr>
<td>MF=(E,stor addr,COMPLETE)</td>
<td>Default: COMPLETE</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the GETDSAB macro, with the following exception:

- **MF=(E,stor addr)**
- **MF=(E,stor addr,COMPLETE)**
  - Specifies the execute form of the macro.
  - The `stor addr` parameter specifies the name of a required storage area for the parameter list.
  - The COMPLETE parameter specifies the degree of macro parameter syntax checking. COMPLETE checks for required macro keywords and supplies defaults for optional parameters that are not specified.
Chapter 18. GETMAIN — Allocate virtual storage

Description

Use the GETMAIN macro to request one or more areas of virtual storage.

Before obtaining storage, be sure to read the topic about selecting the right subpool for virtual storage requests in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

You can also use the STORAGE macro to obtain storage. Compared to GETMAIN, STORAGE provides an easier-to-use interface and has fewer restrictions and locking requirements. See the virtual storage management chapter in [z/OS MVS Programming: Authorized Assembler Services Guide](#) for a comparison of GETMAIN and STORAGE.

The GETMAIN macro is also described in [z/OS MVS Programming: Assembler Services Reference ABE-HSP](#) with the exception of the BRANCH and OWNER parameters.

Note:

1. When you obtain storage, the system clears the requested storage to zeros if you obtain either:
   - 8192 bytes or more from a pageable, private storage subpool.
   - 4096 bytes or more from a pageable, private storage subpool, with BNDRY=PAGE specified.
   In all other cases you must not assume that the storage is cleared to zeros.
   The caller can specify CHECKZERO=YES to detect these and other cases where the system clears the requested storage to zeros.

2. Do not allocate user key (8-15) storage in the common area because it can be read or written by any program in any address space.

The GETMAIN macro provides two types of entry linkage: SVC entry and branch entry. If you do not specify the BRANCH parameter, the GETMAIN service receives control through **SVC entry**. If you specify the BRANCH parameter, the GETMAIN service receives control through **branch entry**.

If you use GETMAIN to request real storage backing above 2 gigabytes, but your system does not support 64-bit storage, your request will be treated as a request for backing above 16 megabytes, even on earlier releases of z/OS that do not support backing above 2 gigabytes. However, boundary requirements indicated by the CONTBDY and STARTBDY parameters will be ignored by earlier releases of z/OS.
GETMAIN macro

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
</table>

For subpools 131 and 132, **one or more of the following**:
- Supervisor state
- PSW key 0-7
- APF-authorization.
- PSW key mask (PKM) that allows the calling program to switch its PSW key to match the key of the storage to be obtained.

For other subpools: **one or more of the following**:
- Supervisor state
- PSW key 0-7
- APF-authorization.

**Dispatchable unit mode:**
- For SVC entry: task.
- For branch entry: supervisor state and PSW key 0.

**Cross memory mode:**
- For SVC entry: PASN=HASN=SASN.
- For branch entry: any PASN, any HASN, any SASN.

**AMODE:**
- For SVC entry: 24- or 31- or 64-bit.
- For branch entry: 24- or 31-bit.
  - For R, LC, LU, VC, VU, EC, or EU requests: If the calling program is in 31-bit mode, the system treats all addresses and values as 31-bit. Otherwise, the system treats addresses and values as 24-bit.
  - For RC, RU, VRC, and VRU requests: The system treats all addresses and values as 31-bit.

**ASC mode:**
- For BRANCH=(YES,GLOBAL): primary or access register (AR).
- For all other requests: primary.

Callers in AR mode must use BRANCH=(YES,GLOBAL) and can obtain only global (common) storage.

**Interrupt status:**
- For BRANCH=(YES,GLOBAL): disabled for I/O and external interrupts.
- For all other requests: enabled for I/O and external interrupts.

**Locks:**
- For SVC entry: no locks may be held.
- For BRANCH=YES: your program must hold the local lock for the currently addressable address space. This must be the address space from which the storage is to be obtained.
- For BRANCH=YES, when running in cross-memory mode: your program must hold the CML lock for the currently addressable address space. This must be the address space from which the storage is to be obtained.
- For BRANCH=(YES,GLOBAL): your program must be in an z/OS-recognized state of disablement, which can be attained by obtaining the CPU lock.
Environmental factor Requirement
Control parameters:
For LC, LU, VC, VU, EC, EU requests: control parameters must be in the primary address space.
For other requests: control parameters are in registers.

Programming requirements
Before issuing the GETMAIN macro in AR mode, issue SYSSTATE ASCENV=AR.

Restrictions
• For SVC entry, the caller cannot have an EUT FRR established.

Input register information for SVC entry
Before issuing the GETMAIN macro without the BRANCH parameter (SVC entry)
the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information for SVC entry
For LC, LU, VC, VU, EC, and EU requests: when control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Contains the return code.</td>
</tr>
</tbody>
</table>

For RC, RU, and R requests: when control returns to the caller the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Contains the return code.</td>
</tr>
</tbody>
</table>

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

For VRC and VRU requests: when control returns to the caller the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>For a successful request, contains the length of the storage obtained. Otherwise, used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.</td>
</tr>
</tbody>
</table>

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).
GETMAIN macro

15 Contains the return code.

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Input register information for BRANCH=YES

Before issuing the GETMAIN macro with BRANCH=YES, the caller must ensure that the following GPRs contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>The address of the input TCB, if you are obtaining private storage. Set GPR 4 to 0 or the address of a TCB in the currently addressable address space. Setting the GPR 4 to 0 identifies the input TCB as the TCB that owns the cross-memory resources for the currently addressable address space (task whose TCB address is in ASCBXTCB). For an explanation of the term input TCB, and to determine system-assigned defaults for private storage ownership, see the topic about selecting the right subpool for virtual storage requests in z/OS MVS Programming: Authorized Assembler Services Guide.</td>
</tr>
<tr>
<td>7</td>
<td>The address of the ASCB for the currently addressable address space.</td>
</tr>
</tbody>
</table>

Output register information for BRANCH=YES

For RC, RU, and R requests: when control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system. Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).</td>
</tr>
<tr>
<td>2</td>
<td>Unchanged</td>
</tr>
<tr>
<td>3</td>
<td>For R requests, unchanged. For RC and RU requests, used as a work register by the system.</td>
</tr>
<tr>
<td>4-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Contains the return code.</td>
</tr>
</tbody>
</table>

For VRC and VRU requests: when control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>For a successful request, contains the length of the storage obtained. Otherwise, used as a work register by the system. storage obtained. Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).</td>
</tr>
<tr>
<td>1</td>
<td>The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.</td>
</tr>
</tbody>
</table>
GETMAIN macro

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-4</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>5-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Contains the return code.</td>
</tr>
</tbody>
</table>

For EC, EU, LC, LU, VC, and VU requests: when control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-4</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>5-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Contains the return code.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-4</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>5-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

Input register information for BRANCH=(YES,GLOBAL)

For RC, RU, VRC, and VRU requests (the only valid requests with BRANCH=(YES,GLOBAL)): the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information for BRANCH=(YES,GLOBAL)

For RC and RU requests: when control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.</td>
</tr>
</tbody>
</table>

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>3-4</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>5-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Contains the return code.</td>
</tr>
</tbody>
</table>

For VRC and VRU requests: when control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>For a successful request, contains the length of the storage obtained. Otherwise, used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.</td>
</tr>
</tbody>
</table>
GETMAIN macro

**Note:** A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Unchanged</td>
</tr>
<tr>
<td>3-4</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>5-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Contains the return code.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the service returns control.

**Performance implications**

Repeatedly issuing the GETMAIN macro can slow down performance. If your program requires many identically sized storage areas, use the CPOOL macro or callable cell pool services for better performance.

**Syntax**

The standard form of the GETMAIN macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em> symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede GETMAIN.</td>
</tr>
<tr>
<td>GETMAIN</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow GETMAIN.</td>
</tr>
<tr>
<td>LC,LA= length addr, A= addr</td>
<td><em>length addr</em>: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>LU,LA= length addr, A= addr</td>
<td><em>length value</em>: symbol, decimal number, or register (2) - (12).</td>
</tr>
<tr>
<td>VC,LA= length addr, A= addr</td>
<td>If RC or RU is specified, register (0)</td>
</tr>
<tr>
<td>VU,LA= length addr, A= addr</td>
<td>may also be specified.</td>
</tr>
<tr>
<td>EC,LV= length value, A= addr</td>
<td><em>addr</em>: A-type address or register (2) - (12).</td>
</tr>
<tr>
<td>EU,LV= length value, A= addr</td>
<td>*Note: RC, RU, VRC, or VRU must be used for address greater than 16 megabytes.</td>
</tr>
<tr>
<td>RC,LV= length value</td>
<td></td>
</tr>
<tr>
<td>RU,LV= length value</td>
<td></td>
</tr>
<tr>
<td>R,LV= length value</td>
<td></td>
</tr>
</tbody>
</table>
### GETMAIN macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VRC, LV= (maximum length value, minimum length value)</td>
<td>maximum length value: symbol, decimal number, or register (2) - (12).</td>
</tr>
<tr>
<td>VRU, LV= (maximum length value, minimum length value)</td>
<td>minimum length value: symbol, decimal number, or register (2) - (12).</td>
</tr>
<tr>
<td>SP=subpool nmbr</td>
<td>subpool nmbr: symbol or decimal number 0-255; or register (2) - (12).</td>
</tr>
<tr>
<td>,BNDRY=DBLWD</td>
<td>Default: BNDRY=DBLWD</td>
</tr>
<tr>
<td>,BNDRY=PAGE</td>
<td>Note: This parameter may not be specified with R above.</td>
</tr>
<tr>
<td>,CONTBDY=containing_bdy</td>
<td>containing_bdy: Decimal number 3-31 or register (2) - (12).</td>
</tr>
<tr>
<td>,STARTBDY=starting_bdy</td>
<td>starting_bdy: Decimal number 3-31 or register (2) - (12).</td>
</tr>
<tr>
<td>,BRANCH=YES</td>
<td>Note: BRANCH=(YES,GLOBAL) may be specified only with RC, RU, VRC, or VRU.</td>
</tr>
<tr>
<td>,BRANCH=(YES,GLOBAL)</td>
<td></td>
</tr>
<tr>
<td>,KEY=key number</td>
<td>key number: decimal numbers 0-15, or register (2) - (12).</td>
</tr>
<tr>
<td>,LOC=24</td>
<td>Note: This parameter can only be used with RC, RU, VRC, or VRU. On all other forms, LOC=24 is used.</td>
</tr>
<tr>
<td>,LOC=(24,31)</td>
<td></td>
</tr>
<tr>
<td>,LOC=(24,64)</td>
<td></td>
</tr>
<tr>
<td>,LOC=31</td>
<td></td>
</tr>
<tr>
<td>,LOC=(31,31)</td>
<td></td>
</tr>
<tr>
<td>,LOC=(31,64)</td>
<td></td>
</tr>
<tr>
<td>,LOC=RES</td>
<td>Default: LOC=RES</td>
</tr>
<tr>
<td>,LOC=(RES,31)</td>
<td></td>
</tr>
<tr>
<td>,LOC=(RES,64)</td>
<td></td>
</tr>
<tr>
<td>,LOC=EXPLICIT</td>
<td>Note: You must specify the INADDR parameter with</td>
</tr>
<tr>
<td>,LOC=(EXPLICIT,24)</td>
<td>EXPLICIT.</td>
</tr>
<tr>
<td>,LOC=(EXPLICIT,31)</td>
<td></td>
</tr>
</tbody>
</table>
GETMAIN macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,LOC=(EXPLICIT,64)</td>
<td></td>
</tr>
<tr>
<td>,INADDR=stor addr</td>
<td>stor addr: RX-type address or register (1)-(12).</td>
</tr>
<tr>
<td>Note: This parameter can only be specified with LOC=EXPLICIT.</td>
<td></td>
</tr>
<tr>
<td>,OWNER=HOME</td>
<td>Default: OWNER=HOME</td>
</tr>
<tr>
<td>,OWNER=PRIMARY</td>
<td></td>
</tr>
<tr>
<td>,OWNER=SECONDARY</td>
<td></td>
</tr>
<tr>
<td>,OWNER=SYSTEM</td>
<td></td>
</tr>
<tr>
<td>,CHECKZERO=YES</td>
<td>Default: CHECKZERO=NO</td>
</tr>
<tr>
<td>,CHECKZERO=NO</td>
<td>Note: CHECKZERO may be specified only with RC, RU, VRC, or VRU.</td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>value: Any valid assembler character string</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows.

The first parameter of the GETMAIN macro is positional and is required. This parameter describes the type or mode of the GETMAIN request. The first parameter can be one of the following values:

- **LC, LA=**length addr, A=addr
- **LU, LA=**length addr, A=addr
- **VC, LA=**length addr, A=addr
- **VU, LA=**length addr, A=addr
- **EC, LV=**length value, A=addr
- **EU, LV=**length value, A=addr
- **RC, LV=**length value
- **RU, LV=**length value
- **R, LV=**length value
- **VRC, LV=**(maximum length value, minimum length value)
- **VRU, LV=**(maximum length value, minimum length value)

LC and LU indicate conditional (LC) and unconditional (LU) list requests, and specify requests for one or more areas of virtual storage. The length of each virtual storage area is indicated by the values in a list beginning at the address specified in the LA parameter. The address of each of the virtual storage areas is returned in a list beginning at the address specified in the A parameter. No virtual storage is allocated unless all of the requests in the list can be satisfied.

VC and VU indicate conditional (VC) and unconditional (VU) variable requests, and specify requests for single areas of virtual storage. The length of the single virtual storage area is between the two values at the address specified in the LA parameter. The address and actual length of the allocated virtual storage area are returned by the system at the address indicated in the A parameter.

EC and EU indicate conditional (EC) and unconditional (EU) element requests, and specify requests for single areas of virtual storage. The length of the single
virtual storage area is indicated by the parameter, \( LV=\text{length value} \). The address of the allocated virtual storage area is returned at the address indicated in the A parameter.

RU and R indicate unconditional register requests; RC indicates a conditional register request. RC, RU, and R specify requests for single areas of virtual storage. The length of the single virtual area is indicated by the parameter, \( LV=\text{length value} \). The address of the allocated virtual storage area is returned in register 1.

VRC and VRU indicate variable register conditional (VRC) and unconditional (VRU) requests for a single area of virtual storage. The length returned will be between the maximum and minimum lengths specified by the parameter \( LV=(\text{maximum length value}, \text{minimum length value}) \). The address of the allocated virtual storage is returned in register 1 and the length in register 0.

Note:
1. A conditional request indicates that the active unit of work is not to be abnormally terminated if there is insufficient contiguous virtual storage to satisfy the request. A conditional request does not prevent all abnormal terminations. For example, if the request has incorrect or inconsistent parameters, the system abnormally terminates the active unit of work. An unconditional request indicates that the active unit of work is to be abnormally terminated whenever the request cannot complete successfully.

2. The LC, LU, VC, VU, EC, EU, and R requests can be used only to obtain virtual storage with addresses below 16 megabytes. The RC, RU, VRC, and VRU requests can be used to obtain virtual storage with addresses above 16 megabytes.

LA specifies the virtual storage address of consecutive fullwords starting on a fullword boundary. Each fullword must contain the required length in the low-order three bytes, with the high-order byte set to 0. The lengths should be multiples of 8; if they are not, the system uses the next higher multiple of 8. If VC or VU was coded, two words are required. The first word contains the minimum length required, the second word contains the maximum length. If LC or LU was coded, one word is required for each virtual storage area requested; the high-order bit of the last word must be set to 1 to indicate the end of the list. The list must not overlap the virtual storage area specified in the A parameter.

\( LV=\text{length value} \) specifies the length, in bytes, of the requested virtual storage. The number should be a multiple of 8; if it is not, the system uses the next higher multiple of 8. If R is specified, \( LV=(0) \) may be coded; the low-order three bytes of register 0 must contain the length, and the high-order byte must contain the subpool number. \( LV=(\text{maximum length value}, \text{minimum length value}) \) specifies the maximum and minimum values of the length of the storage request.

The A parameter specifies the virtual storage address of consecutive fullwords, starting on a fullword boundary. The system places the address of the virtual storage area allocated in one or more words. If E was coded, one word is required. If LC or LU was coded, one word is required for each entry in the LA list. If VC or VU was coded, two words are required. The first word contains the address of the virtual storage area, and the second word contains the length actually allocated. The list must not overlap the virtual storage area specified in the LA parameter.
GETMAIN macro

\( \text{SP=\text{subpool nmbr}} \)

Specifies the number of the subpool from which the virtual storage area is to be allocated. If you specify a register, the subpool number must be in bits 24-31 of the register, with bits 0-23 set to zero. Valid subpool numbers range from 0 to 255. See the topic about selecting the right subpool for virtual storage requests in [z/OS MVS Programming: Authorized Assembler Services Guide](https://www.ibm.com) for detailed guidance on subpool selection.

**Note:**

1. Callers running in supervisor state and key zero, who specify subpool 0, will obtain storage from subpool 252. Therefore, when requesting a dump of this storage using the SDUMP or SDUMPX macro, they must specify subpool 252 rather than 0.
2. Requests for storage from subpools 240 and 250 are translated to subpool 0 storage requests.

\( \text{BNDRY=DBLWD} \)

\( \text{BNDRY=PAGE} \)

Specifies that alignment on a doubleword boundary (DBLWD) or alignment with the start of a virtual page on a 4K boundary (PAGE) is required for the start of a requested area.

If the request specifies one of the LSQA or SQA subpools, the system ignores the BNDRY=PAGE keyword. Requests for storage from these subpools are then fulfilled from a single page, unless the request is greater than a page. See the virtual storage management chapter in [z/OS MVS Programming: Authorized Assembler Services Guide](https://www.ibm.com) for a list of LSQA and SQA subpools.

\( \text{CONTBDY=containing_bdy} \)

Specifies the boundary the obtained storage must be contained within. Specify a power of 2 that represents the containing boundary. Supported values are 3-31. For example, CONTBDY=10 means the containing boundary is \(2^{10}\), or 1024 bytes. The containing boundary must be at least as large as the maximum requested boundary. The obtained storage will not cross an address that is a multiple of the requested boundary.

If a register is specified, the value must be in bits 24-31 of the register. CONTBDY is valid only with RC or RU.

CONTBDY is not valid with LOC=EXPLICIT or BNDRY=PAGE.

CONTBDY applies to all subpools.

Generally, if you omit this parameter, there is no containing boundary. However, if the GETMAIN is for SQA or LSQA, and is for less than 4 KB, and STARTBDY is specified, the default of CONTBDY is 12, ensuring that the GETMAIN stays within a 4 KB page boundary.

For GETMAIN macros that specify a CONTBDY parameter value that is larger than 12, it is possible that the allocated area spans across a 4 KB page boundary, even when the area is less than or equal to 4 KB and in an SQA or LSQA subpool.

\( \text{STARTBDY=starting_bdy} \)

Specifies the boundary the obtained storage must start on. Specify a power of 2 that represents the start boundary. Supported values are 3-31. For example, STARTBDY=10 means the start boundary is \(2^{10}\), or 1024 bytes. The obtained storage will begin on an address that is a multiple of the requested boundary.

If a register is specified, the value must be in bits 24-31 of the register. STARTBDY is valid only with RC or RU.
GETMAIN macro

STARTBDY is not valid with LOC=EXPLICIT or BNDRY=PAGE.
STARTBDY applies to all subpools.
If you omit this parameter, the start boundary is 8 bytes (equivalent to specifying STARTBDY=3).

\textbf{,BRANCH=YES, BRANCH=(YES,GLOBAL)}

Specifies that a branch entry is to be used.

BRANCH=YES allows both local (private) and global (common) storage to be allocated. See \textit{Input register information for BRANCH=YES} on page 190 for specific information on input register requirements.

BRANCH=(YES,GLOBAL) allows only global storage to be allocated. With BRANCH=(YES,GLOBAL), the SP parameter may designate only subpools 226-228, 231, 239, 241, 245, 247, or 248. BRANCH=(YES,GLOBAL) is valid only with RC, RU, VRC, or VRU.

\textbf{,KEY=key number}

Specifies the storage key in which the storage is to be obtained. The valid storage keys are 0-15. If a register is specified, the storage key must be in bits 24-27 of the register. KEY is valid with RC, RU, VRC, or VRU, and applies to subpools 129-132, 227-231, 241, and 249. If you specify KEY without specifying RC, RU, VRC, or VRU, or use KEY for any other subpools, the system ignores the KEY parameter. BRANCH is required with KEY for subpools 227-231, 241, and 249. BRANCH=(YES,GLOBAL) is not valid for subpools 129-132, 229-230, and 249. See the virtual storage management chapter in \textit{z/OS MVS Programming: Authorized Assembler Services Guide} for information about how the system determines the storage key to assign to your storage request.

\textbf{,LOC=24, LOC=(24,31), LOC=(24,64), LOC=31, LOC=(31,31), LOC=(31,64), LOC=RES, LOC=(RES,31), LOC=(RES,64), LOC=EXPLICIT, LOC=(EXPLICIT,24), LOC=(EXPLICIT,31), LOC=(EXPLICIT,64)}

Specifies the location of virtual storage and central (also called real) storage. This is especially helpful for callers with 24-bit dependencies. When LOC is specified, central storage is allocated anywhere until the storage is fixed, (for example, using the PGSER macro). You can specify the location of central storage (after the storage is fixed) and virtual storage (whether or not the storage is fixed) using the following LOC parameter values:

LOC=24 indicates that central and virtual storage are to be located below 16 megabytes.

\textbf{Note:}

1. Specifying LOC=BELOW is the same as specifying LOC=24. LOC=BELOW is still supported, but IBM recommends using LOC=24 instead.
2. LOC=24 should not be used to allocate disabled reference (DREF) storage. If issued in AMODE24, an abend B78 will result. In AMODE31, the LOC=24 parameter will be ignored, and the caller will be given an address above 16 megabytes.

3. For GETMAINs from all SQA subpools, central storage will sometimes be above 16 mg even when LOC=24 or LOC=BELLOW is coded.

LOC=(24,31) indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere below 2 gigabytes.

**Note:** Specifying LOC=(BELLOW,ANY) is the same as specifying LOC=(24,31). LOC=(BELLOW,ANY) is still supported, but IBM recommends using LOC=(24,31) instead.

LOC=(24,64) indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere in 64-bit storage.

LOC=31 and LOC=(31,31) indicate that virtual and central storage can be located anywhere below 2 gigabytes.

**Note:** Specifying LOC=ANY or LOC=(ANY,ANY) is the same as specifying LOC=31 or LOC=(31,31). LOC=ANY and LOC=(ANY,ANY) are still supported, but IBM recommends using LOC=31 or LOC=(31,31) instead.

LOC=(31,64) indicates that virtual storage is to be located below 2 gigabytes and central storage can be located anywhere in 64-bit storage.

**Note:** When you specify LOC=31, the actual location of the virtual storage (that is, whether it is above or below 16 megabytes) depends on the subpool you specify on the SP parameter:

- Some subpools (for example, 203-204) are supported only above 16 megabytes. For these subpools, GETMAIN locates virtual storage above 16 megabytes. If you specify LOC=24 for one of these subpools, the system abends your program.

All other subpools are supported both above and below 16 megabytes. For these subpools, specifying LOC=31 causes GETMAIN to try to allocate virtual storage above 16 megabytes. If the attempt fails, GETMAIN tries to allocate virtual storage below 16 megabytes. If this attempt also fails, GETMAIN does not allocate any storage.

All other subpools are supported both above and below 16 megabytes. For these subpools, specifying LOC=31 causes GETMAIN to try to allocate virtual storage above 16 megabytes. If the attempt fails, GETMAIN tries to allocate virtual storage below 16 megabytes. If this attempt also fails, GETMAIN does not allocate any storage.

When you use LOC=RES to allocate storage that can reside either above or below 16 megabytes, LOC=RES indicates that the location of virtual and central storage depends on the location of the caller. If the caller resides below 16 megabytes, virtual and central storage are to be located below 16 megabytes. If the caller resides above 16 megabytes, virtual and central storage are to be located either above or below 16 megabytes.

LOC=(RES,31) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere below 2 gigabytes. In either case, central storage can be located anywhere below 2 gigabytes.
Note: Specifying LOC=(RES,ANY) is the same as specifying LOC=(RES,31).
LOC=(RES,ANY) is still supported, but IBM recommends using LOC=(RES,31) instead.

LOC=(RES,64) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere in 31-bit storage. In either case, central storage can be located anywhere in 64-bit storage.

Note: If your program resides below 16 megabytes but runs with 31-bit addressing mode, you can specify LOC=RES (as a default or explicitly) or LOC=(RES,31) to obtain storage from a subpool supported only above 16 megabytes. Do not specify subpools supported only above 16 megabytes on requests using LOC=RES or LOC=(RES,31) if your program resides below 16 megabytes and runs with 24-bit addressing.

LOC=EXPLICIT, LOC=(EXPLICIT,24), LOC=(EXPLICIT,31), or LOC=(EXPLICIT,64) specify that the requested virtual storage is to be located at the address specified with the INADDR parameter, which is required with EXPLICIT. EXPLICIT is valid only for subpools 0-127, 129-132, 240, 250, 251, and 252. You can use LOC=EXPLICIT only with RC or RU. You cannot specify the BNDRY or OWNER parameters with EXPLICIT.

Note: Specifying LOC=(EXPLICIT,BELOW) is the same as specifying LOC=(EXPLICIT,24). Specifying LOC=(EXPLICIT,ANY is the same as specifying LOC=(EXPLICIT,31). The older specifications are still supported, but IBM recommends using the newer specifications instead.

LOC=(EXPLICIT,31) indicates that virtual storage is to be located at the address specified on the INADDR parameter, and central storage can be located anywhere below 2 gigabytes.

LOC=(EXPLICIT,24) indicates that virtual storage is to be located at the address specified on the INADDR parameter, and central storage is to be located below 16 megabytes. The virtual storage address specified on the INADDR parameter must be below 16 megabytes.

LOC=EXPLICIT and LOC=(EXPLICIT,64) indicate that virtual storage is to be located at the address specified on the INADDR parameter, and central storage can be located anywhere in 64-bit storage.

When you specify EXPLICIT on a request for storage from the same virtual page as previously requested storage, you must request it in the same key, subpool, and central storage area as on the previous storage request. For example, if you request virtual storage backed with central storage below 16 megabytes, any subsequent requests for storage from that virtual page must be specified as LOC=(EXPLICIT,24).

\textbf{INADDR=stor addr}

Specifies the desired virtual address for the storage to be obtained. When you specify INADDR, you must specify EXPLICIT on the LOC parameter.

\textbf{Note:}

1. The address specified on INADDR must be on a doubleword boundary.
2. Make sure that the virtual storage address specified on INADDR and the central storage backing specified on the LOC=EXPLICIT parameter are a valid combination. For example, if the address specified on INADDR is for
GETMAIN macro

virtual storage above 16 megabytes, specify LOC=EXPLICIT or LOC=(EXPLICIT,ANY). Valid combinations include:

- Virtual above, central any
- Virtual any, central any
- Virtual below, central below
- Virtual below, central any

,OWNER=HOME
,OWNER=PRIMARY
,OWNER=SECONDARY
,OWNER=SYSTEM

Specifies the entity to which the system will assign ownership of requested CSA, ECSA, SQA, and ESQA storage. The system uses this ownership information to track the use of CSA, ECSA, SQA and ESQA storage. This parameter can have one of the following values:

HOME
    The home address space.
PRIMARY
    The primary address space.
SECONDARY
    The secondary address space.
SYSTEM
    The system (the storage is not associated with an address space); specify this value if you expect the requested storage to remain allocated after termination of the job that obtained the storage.

The default value is OWNER=HOME. The system ignores the OWNER parameter unless you specify a CSA, SQA, ECSA, or ESQA subpool on the SP parameter. The OWNER parameter is valid only on the VC, VU, RC, RU, VRC, and VRU types of GETMAIN requests.

Programs that issue the GETMAIN macro with the OWNER parameter can run on any z/OS system.

,CHECKZERO=YES
,CHECKZERO=NO

Specifies whether or not the return code for a successful completion should indicate if the system has cleared the requested storage to zeroes. When CHECKZERO=NO is specified or defaulted, the return code for a successful completion is 0. When CHECKZERO=YES is specified, the return code for a successful completion is X'14' if the system has cleared the requested storage to zeroes, and 0 if the system has not cleared the requested storage to zeroes.

There is no performance cost to specifying CHECKZERO=YES.

Programs that issue the GETMAIN macro with the CHECKZERO parameter can run on any z/OS system. On a down-level system, CHECKZERO will be ignored, and the return code for a successful completion (conditional or unconditional) will be 0.

,RELATED=value

Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid assembler character string.

ABEND codes

Abend codes the GETMAIN macro might issue are listed below in hexadecimal. For detailed abend code information, see z/OS MVS System Codes.
GETMAIN macro

- 104
- 10A
- 178
- 204
- 20A
- 278
- 30A
- 378
- 40A
- 478
- 504
- 604
- 704
- 70A
- 778
- 804
- 80A
- 878
- 90A
- 978
- A0A
- A78
- B04
- B0A
- B78
- D04
- D0A
- D78

**Return and reason codes**

When the GETMAIN macro returns control to your program and you specified a conditional request, GPR 15 contains one of the following hexadecimal return codes:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Meaning: Successful completion. CHECKZERO=YES was not specified, or the system has not cleared the requested storage to zeroes.</td>
<td>None.</td>
</tr>
</tbody>
</table>
Table 22. Return Codes for the GETMAIN Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 4           | If you did not specify EXPLICIT on the LOC parameter:  
               • **Meaning**: Environmental or system error. Virtual storage was not obtained because insufficient storage is available.  
               • **Action**: If the request was for low private (local) storage, consult the system programmer to see if you have exceeded an installation-determined private storage limit. 
               If the request is for common (global) storage, your system is probably experiencing a common storage shortage and your request cannot be satisfied until the shortage is corrected.  
               
               If you specified EXPLICIT on the LOC parameter:  
               • **Meaning**: Program error. Virtual storage was not obtained because part of the requested storage area is outside the bounds of the user region.  
               • **Action**: Determine why your program is mistakenly requesting storage outside the user region. If the request was for low private (local) storage, consult the system programmer to see if you have exceeded an installation-determined private storage limit. |
| 8           | **Meaning**: System error. Virtual storage was not obtained because the system has insufficient central storage to back the request.  
               **Action**: Report the problem to the system programmer so the cause of the problem can be determined and corrected. |
| C           | **Meaning**: System error. Virtual storage was not obtained because the system cannot page in the page table associated with the storage to be allocated.  
               **Action**: Report the problem to the system programmer so the cause of the problem can be determined and corrected. |
| 10          | **Meaning**: Program error. Virtual storage was not obtained for one of the following reasons: This reason code applies only to GETMAIN requests with LOC=EXPLICIT specified.  
               • Part of the requested area is allocated already.  
               • Virtual storage was already allocated in the same page as this request, but one of the following characteristics of the storage was different:  
                 – The subpool  
                 – The key  
                 – Central storage backing  
               **Action**: Determine why your program is attempting to obtain allocated storage or why your program is attempting to obtain virtual storage with different attributes from the same page of storage. Correct the coding error. |
| 14          | **Meaning**: Successful completion. The system has cleared the requested storage to zeroes. This return code occurs only when CHECKZERO=YES is specified.  
               **Action**: None. |
Example 1
Obtain 400 bytes of storage from subpool 10. If the storage is available, the address will be returned in register 1 and register 15 will contain 0; if storage is not available, register 15 will contain 4.
GETMAIN RC, LV=400, SP=10

Example 2
Obtain 48 bytes of storage from default subpool 0. If the storage is available, the address will be stored in the word at AREAADDR; if the storage is not available, the task will be abnormally terminated.
GETMAIN EU, LV=48, A=AREAADDR

AREAADDR DS F

Example 3
Obtain a minimum of 1024 bytes to a maximum of 4096 bytes of virtual storage from default subpool 0 with virtual and central storage locations either above or below 16 megabytes. If the storage is available, the starting address is to be returned in register 1 and the length of the storage allocated is to be returned in register 0; if the storage is not available, the caller is to be terminated.
GETMAIN VRU, LV=(4096, 1024), LOC=ANY

Example 4
Obtain 248 bytes of storage from subpool 0 using branch entry. To obtain storage from subpool 0, a supervisor state and PSW key 0 caller must specify subpool 240 or 250. If the storage cannot be obtained, the caller is abnormally terminated.
GETMAIN EU, LV=248, A=AREAADDR, BRANCH=YES, SP=250.

Example 5
Obtain 4096 bytes of storage from CSA subpool 231. Assign the storage area storage key 2. Indicate that the system is to assign the storage to the primary address space. If the storage cannot be obtained, do not abnormally terminate the caller.
GETMAIN RC, LV=4096, SP=231, BRANCH=(YES, GLOBAL), BNDRY=PAGE, KEY=2, OWNER=PRIMARY
GETMAIN macro
Chapter 19. GQSCAN — Extract information from global resource serialization queue

Description

Use the GQSCAN macro to obtain the status of resources and requestors of those resources. The GQSCAN macro allows you to obtain resource information from the system.

ISGQUERY is the IBM recommended replacement for the GQSCAN service.

The ISGRIB macro allows you to interpret the data that the GQSCAN service routine returns to the user-specified area. The ISGRIB macro maps the resource information block (RIB) and the resource information block extent (RIBE) as shown in z/OS MVS Data Areas in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/

There are two fields in the RIB that you can use to determine whether any RIBEs were not returned:

- RIBTRIBE contains the total number of RIBEs associated with this RIB
- RIBNRIBE contains the total number of RIBEs returned by GQSCAN with this RIB in the user-specified area indicated by the AREA parameter.

Global resource serialization counts and limits the number of outstanding global resource serialization requests. A global resource serialization request is any ENQ, RESERVE, or GQSCAN that causes an element to be inserted into a queue in the global resource serialization request queue area. See “Limiting global resource serialization requests” in Chapter 4: Serialization in z/OS MVS Programming: Authorized Assembler Services Guide.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state with any PSW key. For the SCOPE=GLOBAL and SCOPE=LOCAL parameters, supervisor state.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN=SASN or PASN<del>HASN</del>SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

To interpret the data that the GQSCAN service routine returns in the user-specified area, you must include the ISGRIB mapping macro as a DSECT in your program.
GQSCAN macro

Restrictions
Unauthorized callers of GQSCAN need to be authorized through Security Authorization Facility (SAF) when Multi-level security (MLS) is active. If the caller is not authorized, the request will fail.

When multilevel security support is active on the system, unauthorized callers of ISGQUERY who specify REQINFO=QSCAN must have at least READ authorization to the ISG.QSCANSERVICES.AUTHORIZATION resource in the FACILITY class. You can activate the multilevel security support through the SETROPTS MLACTIVE option in RACF. For general information about defining profiles in the FACILITY class, see z/OS Security Server RACF Command Language Reference and z/OS Security Server RACF Security Administrator’s Guide. For information about multilevel security, see z/OS Planning for Multilevel Security and the Common Criteria.

Input register information
Before issuing the GQSCAN macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

Register
Contents
0  Register 0 contains a fullword reason code if the return code in register 15 is X'0A' or X'0C'. Otherwise, register 0 contains the following two halfword values:
  • The first (high-order) halfword contains the length of the fixed portion of each RIB returned.
  • The second (low-order) halfword contains the length of each RIBE returned or reason code.
1  Contains the number of RIBs that were copied into the area provided
2-13  Unchanged
14  Used as a work register by the system
15  Return code

When control returns to the caller, the ARs contain:

Register
Contents
0-1  Used as work registers by the system
2-13  Unchanged
14-15  Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
In general, the narrower the search parameters (particularly QNAME and RNAME), the less time it takes. Using both a specific QNAME and a specific RNAME gives better performance than using generic prefix.
The use of XSYS=YES (the default) might greatly degrade the performance of the request. Use it only when required.

Polling for ENQ contention through GQSCAN or ISGQUERY is not recommended. See the z/OS MVS Planning: Global Resource Serialization and z/OS MVS Programming: Authorized Assembler Services Guide for more information about monitoring contention through ENF 51.

When you specify SCOPE=GLOBAL, or SCOPE=LOCAL, the performance of programs that issue ENQ, DEQ, or the RESERVE macro might be temporarily degraded while the GQSCAN service is running.

**Syntax**

The standard form of the GQSCAN macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede GQSCAN.</td>
</tr>
<tr>
<td>GQSCAN</td>
<td>One or more blanks must follow GQSCAN.</td>
</tr>
<tr>
<td>AREA=(area addr,area size)</td>
<td>area addr: A-type address or register (2) - (12). area size: symbol, decimal digit, or register (2) - (12). Note: AREA cannot be specified with QUIT=YES.</td>
</tr>
<tr>
<td>,REQLIM=value</td>
<td>value: symbol, decimal digit, register (2) - (12), or the word MAX.</td>
</tr>
<tr>
<td>,REQLIM=MAX</td>
<td>Default: REQLIM=MAX</td>
</tr>
<tr>
<td>,SCOPE=ALL</td>
<td>Default: SCOPE=STEP</td>
</tr>
<tr>
<td>,SCOPE=STEP</td>
<td></td>
</tr>
<tr>
<td>,SCOPE=SYSTEM</td>
<td></td>
</tr>
<tr>
<td>,SCOPE=SYSTEMS</td>
<td></td>
</tr>
<tr>
<td>,SCOPE=LOCAL</td>
<td></td>
</tr>
<tr>
<td>,SCOPE=GLOBAL</td>
<td></td>
</tr>
<tr>
<td>,RESERVE=YES</td>
<td>Default: All resources requested with RESERVE and all resources requested with ENQ.</td>
</tr>
<tr>
<td>,RESERVE=NO</td>
<td>resources requested with ENQ.</td>
</tr>
<tr>
<td>,RESNAME=(qname addr, rname addr, rname length), [GENERIC</td>
<td>SPECIFIC]</td>
</tr>
</tbody>
</table>
### Parameters

The parameters are explained as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>qname length()</code></td>
<td>Default: qname length of eight.</td>
</tr>
<tr>
<td><code>,SYSNAME=(sysname addr)</code></td>
<td><code>sysname addr</code>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>[asid value]</code></td>
<td><code>asid value</code>: symbol, decimal digit, or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: Provide <code>rname addr</code> only when <code>qname addr</code> is used. Code <code>rname length</code> if a register is specified for <code>rname addr</code>. Code an <code>asid value</code> only when the <code>sysname addr</code> is used.</td>
</tr>
<tr>
<td><code>,QUIT=YES</code></td>
<td>Default: QUIT=NO</td>
</tr>
<tr>
<td><code>,QUIT=NO</code></td>
<td><strong>Note</strong>: QUIT=YES is mutually exclusive with all parameters but TOKEN and MF.</td>
</tr>
<tr>
<td><code>,REQCNT=value</code></td>
<td><code>value</code>: decimal digit or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td><strong>Default</strong>: REQCNT=0</td>
</tr>
<tr>
<td><code>,OWNERCT=value</code></td>
<td><code>value</code>: decimal digit or register (2) - (12).</td>
</tr>
<tr>
<td><code>,WAITCNT=value</code></td>
<td><code>value</code>: decimal digit or register (2) - (12).</td>
</tr>
<tr>
<td><code>,TOKEN=addr</code></td>
<td><code>addr</code>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,XSYS=YES</code></td>
<td>Default: XSYS=YES</td>
</tr>
<tr>
<td><code>,XSYS=NO</code></td>
<td><strong>Note</strong>: XSYS=NO is mutually exclusive with TOKEN, QUIT=YES and SYSNAME, when SYSNAME is not equal to zero or zero and the asid value(0,asid value). In a global resource serialization ring complex, XSYS=NO is ignored.</td>
</tr>
</tbody>
</table>

### AREA=(area addr,area size)

Specifies the location and size of the area where information extracted from the global resource serialization resource queues is to be placed. The minimum size is the amount needed to describe a single resource, which is the length of the fixed portions of the RIB and the maximum size rounded up to a fullword value. IBM recommends that you use a minimum of 1024 bytes as the area size.

### REQLIM=value

Specifies the maximum number of owners and waiters to be returned for each individual resource within the specification of RESNAME, which can be any value between 0 and 2^35-1. MAX specifies 2^35-1 (32767).
Specifies that you want information only for resources having the indicated scope. STEP, SYSTEM, or SYSTEMS is the scope specified on the resource request. If you specify SCOPE=ALL (meaning STEP, SYSTEM, and SYSTEMS), the system returns information for all resources the system recognizes that have the specified RESNAME, RESERVE, or SYSNAME characteristics. If you specify SCOPE=LOCAL, information is returned about this system's resources that are not being shared with other systems in the global resource serialization complex. If you specify SCOPE=GLOBAL, information is returned about resources that are being shared with other systems in the global resource serialization complex. Remember that entries in the resource name lists can cause the scope to change.

If you specify RESERVE=YES, information is only returned for the requestors of the resource, that requested the resource with the RESERVE macro. If, for example, the resource also had requestors with the ENQ macro, the ENQ requestor's information would not be returned for the resource.

RESERVE=NO information is only returned for the requestors of the resource that requested the resource with the ENQ macro. In other words, if the resource also had requestors with the RESERVE macro, the RESERVE requestor's information would not be returned for the resource.

RESNAME identifies an individual resource or group of resources that GQSCAN will examine.

RESNAME with (rname) indicates the name of one resource.

The qname addr specifies the address of the 8-character major name of the requested resource.

The rname addr specifies the virtual storage address of a 1 to 255-byte minor name used with the major name to represent a single resource. Information returned is for a single resource unless you specify SCOPE=ALL, in which case it could be for three resources (STEP, SYSTEM, and SYSTEMS) or SCOPE=LOCAL in which case it could be for two resources (STEP and SYSTEM) if there is a matching name in each of these categories. If the name specified by rname is defined by an EQU assembler instruction, the rname length must be specified.

The rname length specifies the length of the minor name. If you use the register form, specify length in the low-order (rightmost) byte. The length must match the rname length specified on ENQ or RESERVE.

GENERIC specifies that the rname of the requested resource must match but only for the length specified. For example, an ENQ for SYS1.PROCLIB would match the GQSCAN rname specified as SYS1 for an rname length of 4.

SPECIFIC specifies that the rname of the requested resource must exactly match the GQSCAN rname.
Note: GENERIC and SPECIFIC are mutually exclusive.

The qname length specifies the number of characters in a resource qname that must match the GQSCAN qname specified by RESNAME. You must specify a qname length to request a GQSCAN for a generic qname. For example, an ENQ with a qname of SYSDSN would match a GQSCAN specifying GENERIC with a qname of SYSD and qname length of 4. Specify zero for the qname length (with any qname) to request a generic GQSCAN matching any resource qname. If you do not specify a qname length, GQSCAN uses the default of 8.

\texttt{SYSNAME=\{sysname addr [, asid value]\}}

Specify SYSNAME to tell GQSCAN to return information for resources requested by tasks running on the MVS system specified in an 8-byte field pointed to by the address in \texttt{sysname address} and the \texttt{asid value}, a 4-byte address space identifier, right justified. Valid SYSNAMEs are specified in the IEASYxx parmlib member.

Information returned includes only those resources whose \texttt{sysname addr} and \texttt{asid value} match the ones specified. SYSNAME=0 or SYSNAME=(0,asid value), specifies that the system name is that of the system on which GQSCAN is issued. The system issues return code X'0A' with a reason code of X'0C', if SYSNAME≠0 or SYSNAME≠(0,asid value) is specified with XSYS=NO.

\texttt{QUIT=YES}

QUIT=NO indicates that you do not want to end the current global resource serialization queue scan. QUIT=YES tells GQSCAN to stop processing the current global resource serialization queue scan and release the storage allocated to accumulate the information specified in the token.

If you specify QUIT=YES, you must specify the TOKEN parameter. If you specify QUIT=YES without the TOKEN parameter, the system issues abend X'09A'.

If you specify QUIT=YES without the TOKEN parameter, the system issues return code X'0A' with a reason code of X'34'. Specifying QUIT=YES with TOKEN=0 will result in the system issuing return code X'0A' with a reason code of X'2C'.

If you specify QUIT=YES with a token that was previously obtained through GQSCAN with SCOPE=LOCAL or SCOPE=GLOBAL, your program must be in supervisor state when it issues GQSCAN with QUIT=YES.

If you specify QUIT=YES with XSYS=NO, the system issues return code X'0A' with a reason code of X'0C'.

\texttt{REQCNT=rcount}

Specifies that you want GQSCAN to return resource information only when the total number of requesters (owners plus waiters) for an individual resource is greater than or equal to \texttt{rcount}, which can be any value between 0 and $2^{31}-1$.

\texttt{OWNERCT=ocount}

Specifies that you want GQSCAN to return resource information only when the total number of owners for an individual resource is greater than or equal to \texttt{ocount}, which can be any value between 0 and $2^{31}-1$.

\texttt{WAITCNT=wcount}

Specifies that you want GQSCAN to return resource information only when the total number of waiters for an individual resource is greater than or equal to \texttt{wcount}, which can be any value between 0 and $2^{31}-1$. 
**GQSCAN macro**

**OWNERCT=ocount, WAITCNT=wcount**
Specify that you want GQSCAN to return resource information only when the total number of owners for an individual resource is greater than or equal to *ocount* or when the total number of waiters for an individual resource is greater than or equal to *wcount*.

**TOKEN=addr**
Specify the address of a fullword of storage that the GQSCAN service routine can use to provide you with any remaining information in subsequent invocations. If the token value is zero, the scan starts at the beginning of the resource queue. If the token value is not zero, the scan resumes at the point specified on TOKEN. Specify the same token value that GQSCAN returned on its previous invocation to continue where processing left off on the previous invocation.

When providing a non-zero token value, you must specify the same scope that you specified on the GQSCAN request that returned the token.

**XSYS=YES**
**XSYS=NO**
Specify whether GQSCAN should be propagated across systems in the global resource serialization complex, to gather complex-wide information. This parameter is ignored in a global resource serialization ring complex, and for requests that only gather local data.

Specify XSYS=YES if the program requires complex-wide global resource serialization information. The caller might be suspended while the information is being gathered. Do **not** specify or default to XSYS=YES if this condition cannot be tolerated.

Specify XSYS=NO if the program will accept global resource serialization information from this system only. The RIBE data will contain information about requestors from other other systems in the complex only if that information is already available on the GQSCAN caller’s system. Otherwise, RIBE data will be provided only for requests from the GQSCAN caller’s system, and the counts in the RIB will reflect only those requests. This request is always handled without placing the caller’s dispatchable unit into a wait.

**ABEND codes**
See [z/OS MVS System Codes](https://www.ibm.com) for more information about the abend codes.

**Return and reason codes**
When GQSCAN returns control, register 15 contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0</strong></td>
<td><strong>Meaning</strong>: Queue scan processing is complete. Data is now in the area you specified. There is no more data to return.  Action: Process the data.</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td><strong>Meaning</strong>: Queue scan processing is complete. No resources matched your request.  Action: Do not try to process any data; none exists.</td>
</tr>
<tr>
<td><strong>8</strong></td>
<td><strong>Meaning</strong>: The area you specified was filled before queue scan processing completed.  Action: If you specified TOKEN, process the information in the area and issue GQSCAN again, specifying the TOKEN returned to you. If you did not specify TOKEN, specify a larger area or specify a TOKEN.</td>
</tr>
</tbody>
</table>
### Table 23. Return Codes for the GQSCAN Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0A</td>
<td>Meaning: The information you specified to GQSCAN is not valid.</td>
</tr>
<tr>
<td></td>
<td>Action: Take the action indicated by the following hexadecimal reason code found in register 0.</td>
</tr>
<tr>
<td>04</td>
<td>Reason Code Meaning: The caller attempted to use GQSCAN before the global resource serialization (GRS) address space was active.</td>
</tr>
<tr>
<td>08</td>
<td>Reason Code Meaning: The size of the reply area, specified by the AREA parameter, is too small to contain a resource information block (RIB) of maximum size.</td>
</tr>
<tr>
<td>0C</td>
<td>Reason Code Meaning: You specified mutually exclusive arguments (RESERVE=YES, RESERVE=NO, RESNAME=, SYSNAME=, or XSYS=NO) to GQSCAN.</td>
</tr>
<tr>
<td>10</td>
<td>Reason Code Meaning: The caller was holding a local lock other than the GRS local lock when GQSCAN was invoked.</td>
</tr>
<tr>
<td>14</td>
<td>The following conditions, in reference to the RESNAME parameter, was detected by GQSCAN:</td>
</tr>
<tr>
<td></td>
<td>• The qname length was specified with a value greater than eight.</td>
</tr>
<tr>
<td></td>
<td>• The qname length value was specified without the qname addr value.</td>
</tr>
<tr>
<td></td>
<td>• The specific parameter was specified with a rname length value of zero.</td>
</tr>
<tr>
<td></td>
<td>• The rname or rname length was specified without the qname addr value.</td>
</tr>
<tr>
<td>18</td>
<td>Reason Code Meaning: The asid value, for the SYSNAME parameter was specified without the sysname addr value.</td>
</tr>
<tr>
<td>1C</td>
<td>Reason Code Meaning: The REQCNT parameter was specified with either the OWNERCNT or WAITCNT parameters.</td>
</tr>
<tr>
<td>20</td>
<td>Reason Code Meaning: The combination of values specified on the SCOPE parameter is not valid.</td>
</tr>
<tr>
<td>28</td>
<td>Reason Code Meaning: An element in GQSCAN's input parameter list was not in the caller's storage protect key.</td>
</tr>
<tr>
<td>2C</td>
<td>Reason Code Meaning: An invalid token was specified to GQSCAN.</td>
</tr>
<tr>
<td>30</td>
<td>Reason Code Meaning: The GQSCAN caller is not authorized to use the restricted interface (SCOPE=LOCAL or GLOBAL).</td>
</tr>
<tr>
<td>34</td>
<td>Reason Code Meaning: QUIT=YES was specified without the TOKEN parameter.</td>
</tr>
<tr>
<td>38</td>
<td>Reason Code Meaning: The caller held a CMS lock other than CMSEQDQ when GQSCAN was invoked.</td>
</tr>
<tr>
<td>3C</td>
<td>Reason Code Meaning: The caller held a lock that violated the environmental restrictions of a service required by GQSCAN.</td>
</tr>
<tr>
<td>40</td>
<td>Reason Code Meaning: The caller invoked GQSCAN in the service request block (SRB) mode.</td>
</tr>
<tr>
<td>44</td>
<td>Reason Code Meaning: The value specified for the REQLIM parameter was not valid.</td>
</tr>
<tr>
<td>48</td>
<td>Reason Code Meaning: The value specified for the REQCNT parameter was not valid.</td>
</tr>
<tr>
<td>4C</td>
<td>Reason Code Meaning: The value specified for the OWNERCNT parameter was not valid.</td>
</tr>
<tr>
<td>50</td>
<td>Reason Code Meaning: The value specified for the WAITCNT parameter was not valid.</td>
</tr>
<tr>
<td>58</td>
<td>SETROPTS MLACTIVE is in effect, and the program is not authorized to issue GQSCAN. Ensure the program is running authorized, or is associated with a userid with at least READ access to the best fit FACILITY class resource profile of the form ISG.QSCANSERVICES.AUTHORIZATION and that the FACILITY class is SETROPTS RACLISTed.</td>
</tr>
</tbody>
</table>
### GQSCAN macro

#### Table 23. Return Codes for the GQSCAN Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0C                      | **Meaning**: System error. Queue scan encountered an abnormal situation while processing. The information in your area is not meaningful. The reason code in register 0 contains one of the following: 

**Reason Code**

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
</tr>
<tr>
<td>04</td>
</tr>
<tr>
<td>08</td>
</tr>
<tr>
<td>0C</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>14</td>
</tr>
</tbody>
</table>

**Action**: Do not try to process any data; none exists. Retry the request one or more times. |

| 10                      | **Meaning**: Program error. An incorrect SYSNAME was specified as input to queue scan. The information in your area is not meaningful. |

**Action**: Specify a valid SYSNAME on the call to GQSCAN. |

| 14                      | **Meaning**: Environmental error. The area you specified was filled before queue scan processing completed. Your request specified TOKEN, but the limit for the number of concurrent resource requests (ENQ, RESERVE, or GQSCAN) has been reached. The information in your area is valid but incomplete. The scan cannot be resumed. |

**Action**: Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded. |

---

### GQSCAN - List form

The list form of the GQSCAN macro is used to construct a non-executable parameter list. This parameter list, or a copy of it for reentrant programs, can be referred to by the execute form of the GQSCAN macro.

The list form of the GQSCAN macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede GQSCAN.</td>
</tr>
<tr>
<td>GQSCAN</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow GQSCAN.</td>
</tr>
<tr>
<td>AREA=(area addr, area size)</td>
<td>area addr: A-type address.</td>
</tr>
<tr>
<td></td>
<td>area size: symbol, decimal digit.</td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Note:** | 1. This parameter cannot be specified with QUIT=YES.  
2. AREA is required on either the list or the execute form of the macro. |

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>,REQLIM=value</strong></td>
<td>value: symbol, decimal digit or the word MAX.</td>
</tr>
<tr>
<td><strong>,REQLIM=MAX</strong></td>
<td>Default: REQLIM=MAX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>,SCOPE=ALL</strong></td>
<td>Default: SCOPE=STEP</td>
</tr>
<tr>
<td><strong>,SCOPE=STEP</strong></td>
<td></td>
</tr>
<tr>
<td><strong>,SCOPE=SYSTEM</strong></td>
<td></td>
</tr>
<tr>
<td><strong>,SCOPE=SYSTEMS</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>,RESERVE=YES</strong></td>
<td>Default: All resources requested with RESERVE and all</td>
</tr>
<tr>
<td><strong>,RESERVE=NO</strong></td>
<td>resources requested with ENQ.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>,RESNAME=</strong>(qname addr)</td>
<td>qname addr: A-type address.</td>
</tr>
<tr>
<td><strong>addr [rname addr,</strong></td>
<td>rname addr: A-type address.</td>
</tr>
<tr>
<td><strong>rname length]</strong></td>
<td>rname length: decimal digit.</td>
</tr>
<tr>
<td>**[GENERIC</td>
<td>SPECIFIC]**</td>
</tr>
<tr>
<td><strong>qname length)</strong></td>
<td>Default: qname length of eight.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>,SYSNAME=</strong>(sysname addr)</td>
<td>sysname addr: A-type address.</td>
</tr>
<tr>
<td><strong>[asid value]</strong></td>
<td>asid value: symbol, decimal digit.</td>
</tr>
</tbody>
</table>

**Note:** rname addr can be provided only when qname addr is used. rname length must be provided if a register is specified for rname addr. An asid value can be coded only when the sysname addr is used.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>,QUIT=YES</strong></td>
<td>Default: QUIT=NO</td>
</tr>
<tr>
<td><strong>,QUIT=NO</strong></td>
<td>Note: Only TOKEN and MF=L can be specified with QUIT=YES.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **,REQCNT=value** | value: decimal digit.  
Default: REQCNT=0 |

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **,OWNERCT=value,WAITCNT=value** | value: decimal digit.  
,value: decimal digit.  
,value: decimal digit.  
,value: decimal digit. |

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>,TOKEN=addr</strong></td>
<td>addr: RX-type address.</td>
</tr>
</tbody>
</table>
### GQSCAN macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,XSYS=YES</td>
<td>Default: XSYS=YES</td>
</tr>
<tr>
<td>,XSYS=NO</td>
<td>Note: XSYS=NO is mutually exclusive with TOKEN, QUIT=YES and SYSNAME, when SYSNAME is not equal to zero or zero and the asid value(0,asid value). In a global resource serialization ring complex, XSYS=NO is ignored.</td>
</tr>
<tr>
<td>,MF=L</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the GQSCAN macro with the following exception:

, MF=L

Specifies the list form of the GQSCAN macro.

### GQSCAN - Execute form

The execute form of the GQSCAN macro can refer to and modify a remote parameter list built by the list form of the macro. There are no defaults for any of the parameters in the execute form of the macro.

The execute form of the GQSCAN macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede GQSCAN.</td>
</tr>
<tr>
<td>GQSCAN</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow GQSCAN.</td>
</tr>
<tr>
<td>AREA=(area addr,area size)</td>
<td>area addr: RX-type address or register (2) - (12). area size: symbol, decimal digit, or register (2) - (12).</td>
</tr>
<tr>
<td>Note:</td>
<td>1. AREA cannot be specified with QUIT=YES. 2. AREA is required on either the list or the execute form of the macro.</td>
</tr>
<tr>
<td>,REQLIM=value</td>
<td>value: symbol, decimal digit, register (2) - (12), or the word MAX.</td>
</tr>
<tr>
<td>,REQLIM=MAX</td>
<td></td>
</tr>
<tr>
<td>,SCOPE=STEP</td>
<td>Note: SCOPE=LOCAL and SCOPE=GLOBAL cannot be coded on the list form of this macro.</td>
</tr>
</tbody>
</table>
### GQSCAN macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>,SCOPE=ALL</code></td>
<td></td>
</tr>
<tr>
<td><code>,SCOPE=SYSTEM</code></td>
<td></td>
</tr>
<tr>
<td><code>,SCOPE=SYSTEMS</code></td>
<td></td>
</tr>
<tr>
<td><code>,SCOPE=LOCAL</code></td>
<td></td>
</tr>
<tr>
<td><code>,SCOPE=GLOBAL</code></td>
<td></td>
</tr>
<tr>
<td><code>,RESERVED=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>,RESERVED=NO</code></td>
<td></td>
</tr>
<tr>
<td><code>,RESERVED=(qname qname addr)</code></td>
<td>RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,resid resid addr, resid resid addr</code></td>
<td>RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>rname length, rname length</code></td>
<td>Decimal digit, register (2) - (12). Default: assembled length of rname.</td>
</tr>
<tr>
<td>`[GENERIC</td>
<td>SPECIFIC], qname length`</td>
</tr>
<tr>
<td><code>,SYSTYPE=(sysname sysname addr)</code></td>
<td>RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>[,asid value] asid value</code></td>
<td>Symbol, decimal digit, or register (2) - (12).</td>
</tr>
<tr>
<td><code>,QUIT=YES</code></td>
<td>Default: QUIT=NO</td>
</tr>
<tr>
<td><code>,QUIT=NO</code></td>
<td>Note: Only TOKEN and MF=(E, parm list addr) can be specified with QUIT=YES.</td>
</tr>
<tr>
<td><code>,REQCNT=value</code></td>
<td>Value: decimal digit or register (2) - (12). Default: REQCNT=0</td>
</tr>
<tr>
<td><code>,OWNERCT=value, WATCNT=value</code></td>
<td>Value: decimal digit.</td>
</tr>
<tr>
<td><code>,OWNERCT=value</code></td>
<td>Value: decimal digit.</td>
</tr>
<tr>
<td><code>,WATCNT=value</code></td>
<td>Value: decimal digit.</td>
</tr>
<tr>
<td><code>,TOKEN=addr</code></td>
<td>RX-type address of register (2) - (12).</td>
</tr>
<tr>
<td><code>,XSYS=YES</code></td>
<td>Default: XSYS=YES</td>
</tr>
<tr>
<td><code>,XSYS=NO</code></td>
<td>Note: XSYS=NO is mutually exclusive with TOKEN, QUIT=YES and SYSNAME, when SYSNAME is not equal to zero or zero and the asid value(0,asid value). In a global resource serialization ring complex, XSYS=NO is ignored.</td>
</tr>
</tbody>
</table>
### Parameters

The parameters are explained under the standard form of the GQSCAN macro with the following exception:

\[ \text{,MF=(E,} \text{list} \text{ addr)} \]

Specifies the execute form of the GQSCAN macro.

\[ \text{list} \text{ addr} \]

specifies the area that the system uses to contain the parameters.
GQSCAN macro
Chapter 20. GTRACE — GTF trace recording

Description

Use the GTRACE macro to record system or application errors through the generalized trace facility (GTF). The GTRACE macro provides three separate functions, depending on the keyword specified:

- GTRACE TEST indicates whether the operator requested a specific user event.
- GTRACE QUERY indicates how much data GTF can store when a program issues GTRACE DATA.
- GTRACE DATA generates GTF trace records for specific events.

Refer to z/OS MVS Diagnosis: Tools and Service Aids and z/OS Problem Management for information about using GTF.

The following description of the GTRACE macro is divided into three sections, one for each function of the macro. The TEST and QUERY functions have only one form each, while the DATA function has standard, list, and execute forms.

GTRACE TEST

The TEST function of the GTRACE macro indicates whether the operator requested a particular user event in response to the USRP option. The system returns the test result as a return code in register 15.

By issuing GTRACE TEST and checking the return code, you can determine whether you need to subsequently issue GTRACE DATA to write the record. If the return code indicates that tracing has been requested by USRP for the specified user event, then issue GTRACE DATA.

Issuing GTRACE TEST before issuing GTRACE DATA is not necessary but you might find it useful to do so if the processing of your code can benefit from learning whether processing is active for the record type you want to record to the generalized trace facility (GTF) before requesting to do that recording.

When the operator requests GTF prompting for specific event identifiers with the USRP option and your program issues GTRACE DATA, the system records the user trace record only when the event identifier specified on GTRACE DATA was also requested with the USRP option. However, the TEST function is still supported for compatibility with existing programs.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state, any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No requirement</td>
</tr>
</tbody>
</table>
GTRACE macro

Environmental factor
Control parameters: Requirement

Must be in the primary address space and all data must reside in primary address space.

Programming requirements
- Include the CVT and the MCHEAD mapping macros.
- When you code the CVT mapping macro, you must not specify PREFIX=YES.

Restrictions
None.

Input register information
Before issuing the GTRACE TEST macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unchanged</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

Performance implications
None.

Syntax
The TEST function of the GTRACE macro is coded as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede GTRACE.</td>
</tr>
<tr>
<td>GTRACE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow GTRACE.</td>
</tr>
<tr>
<td>TEST=YES</td>
<td></td>
</tr>
</tbody>
</table>
Parameters
The parameters are explained as follows:

TEST=YES
Specifies the test function of the GTRACE macro.

,ID=id
Specifies the event ID for the user event that is to be tested. Decimal event IDs 0 through 1023 (X'3FF') are available for user events. You can specify the ID in decimal or in hexadecimal. Use the expression X'\text{id}' to specify a hexadecimal number.

ABEND codes
None.

Return codes
When GTRACE TEST macro returns control to your program, GPR 15 contains a return code.

Table 24. Return Codes for the GTRACE TEST Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: Tracing has not been requested by USRP for the specified user event. Action: Do not issue a GTRACE DATA request to create your trace record for the specified user event ID.</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: Tracing has been requested by USRP for the specified user event. Action: You may issue a GTRACE DATA request to create your trace record for the specified user event ID.</td>
</tr>
</tbody>
</table>

GTRACE QUERY

The QUERY function of the GTRACE macro determines how much data GTF will accept, and returns the value in the variable or register specified with the MAXLNG parameter. This function is useful when your program must run on different levels of MVS that accept different amounts of trace data in GTRACE DATA.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment factor</td>
<td>Problem state, any PSW key</td>
</tr>
<tr>
<td>Minimum authorization</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>24- or 31- or 64-bit</td>
</tr>
<tr>
<td>AMODE</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O or external interrupts</td>
</tr>
</tbody>
</table>
GTRACE macro

Environmental factor: Requirement
Locks: No requirement
Control parameters: Must be in the primary address space

Programming requirements
None.

Restrictions
None.

Input register information
Before issuing the GTRACE QUERY macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Zero</td>
</tr>
</tbody>
</table>

Performance implications
None.

Syntax
The QUERY function of the GTRACE macro is coded as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: Symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>\b</td>
<td>One or more blanks must precede GTRACE.</td>
</tr>
<tr>
<td>GTRACE</td>
<td></td>
</tr>
<tr>
<td>\b</td>
<td>One or more blanks must follow GTRACE.</td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
</tr>
<tr>
<td>.MAXLNG=addr</td>
<td><em>addr</em>: RX-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained as follows:
GTRACE macro

QUERY
Specifies the query function of the GTRACE macro.

,MAXLNG=addr
Specifies the address in which the maximum amount of GTF data is returned.

ABEND codes
None.

Return codes
The return code from GTRACE QUERY is always zero.

GTRACE DATA
The DATA function of the GTRACE macro records system or problem program data in the GTF trace buffers. GTRACE DATA can trace up to 8192 bytes of data.

Data is written only if you requested the event qualifier (through the USRP option) when you started GTF. Therefore, you can issue the GTRACE DATA without issuing a GTRACE TEST.

In earlier releases, GTRACE DATA writes the record to the GTF data set even if the record’s event ID (EID) is excluded from a USRP list in the GTF trace options. Therefore, you need to issue a GTRACE TEST before you issue GTRACE DATA to determine if data is to be collected for the event qualifier.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state, any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN; all data and parameters must reside in the home address space.</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31- or 64-bit. The caller must be in 31-bit mode for GTRACE to record data above 16 megabytes.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No requirement</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space</td>
</tr>
</tbody>
</table>

Programming requirements
None.

Restrictions
None.

Input register information
Before issuing the GTRACE DATA macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.
GTRACE macro

Output register information
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unchanged</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

Performance implications
None.

Syntax
The standard form of the DATA function of the GTRACE macro is coded as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede GTRACE.</td>
</tr>
<tr>
<td>DATA=addr</td>
<td>addr: RX address or register (2) - (12).</td>
</tr>
<tr>
<td>DATA64=addr</td>
<td></td>
</tr>
<tr>
<td>,LNG=nbr</td>
<td>nbr: Symbol, decimal number, hexadecimal number, or register (2) - (12).</td>
</tr>
<tr>
<td>,ID=id</td>
<td>id: Symbol, decimal number, or hexadecimal number.</td>
</tr>
<tr>
<td>,FID=fidname</td>
<td>fidname: Symbol, decimal number, hexadecimal number, or register (2) - (12).</td>
</tr>
<tr>
<td>,PAGEIN=NO</td>
<td>Default: PAGEIN=NO</td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained as follows:

DATA=addr
### DATA64=addr

Specifies the virtual storage address of the data that is to be recorded.

**Note:** DATA64 can be specified only when running in 64-bit address mode (AMODE).

### LNG=nbr

Specifies the number of data bytes (1 through 8192) to be recorded from the address specified by the DATA parameter. You can specify the number in decimal or in hexadecimal. If the number is hexadecimal, use the expression X'nbr' to specify the number.

**Note:** When you specify LNG, the trace record contains the number of bytes that you specify plus 12 bytes, which is the size of the trace record header. The header consists of a 4-byte ASCB address followed by an 8-byte jobname. Thus, if you specify LNG=8192, the trace record has 8204 (8192+12) bytes.

### ID=id

Specifies the event ID that is to be recorded with the data bytes. Decimal event ids 0 through 1023 (X'3FF') are available for user events. You can specify the ID in decimal or in hexadecimal. Use the expression X'id' to specify a hexadecimal number.

### FID=fidname

Specifies the format appendage that controls the formatting of this record. Formatting occurs when the trace output is processed by GTF trace. The format appendage name is formed by appending the 2-digit FID value to the names AMDUSR, HMDUSR, and IMDUSR. Assign FID values as follows:

- X'00': The record is to be dumped in hexadecimal.
- X'01' to X'50': The record contains user format identifiers.

**Note:** If you code FID without any fidname, or if you omit the FID parameter, the system supplies a default fidname of zero.

### PAGEIN=NO, PAGEIN=YES

Specifies that paged-out user data is to be processed (YES) or not to be processed (NO). To ensure that all user data is traced, specify YES.

### ABEND codes

None.

### Return codes

When GTRACE DATA macro returns control to your program, GPR 15 contains a return code.

#### Table 25. Return Codes for the GTRACE DATA Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | **Meaning:** GTF is active. The data was recorded in GTF trace buffers.  
|                         | **Action:** None.     |
| 04                      | **Meaning:** GTF is not active or not active for this particular event ID. No data was recorded.  
|                         | **Action:** None.     |
GTRACE macro

Table 25. Return Codes for the GTRACE DATA Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08                      | **Meaning:** Program error. The value of the LNG keyword is not valid. It must be a number from 1 through 8192. No data was recorded.  
**Action:** Reissue the macro, specifying a valid amount of trace data to be recorded. |
| 0C                      | **Meaning:** Program error. The value of the DATA keyword is not valid. It does not represent an area of storage that the calling program can refer to. No data was recorded.  
**Action:** Correct the problem and reissue the macro. |
| 10                      | **Meaning:** Program error. The value of the FID keyword is not valid. It must be a number from X'0' through X'FF'. No data was recorded.  
**Action:** Correct the problem and reissue the macro. |
| 18                      | **Meaning:** Environmental condition. All GTF buffers are full. No data was recorded.  
**Action:** None. |
| 1C                      | **Meaning:** Program error. The address of the parameter list for GTF is not valid. The parameter list is not in storage that the caller can refer to, or its format is not valid. No data was recorded.  
**Action:** Correct the problem and reissue the macro. |
| 20                      | **Meaning:** Program error. Some of the data to be recorded was paged out. No data was recorded. This return code is not valid with PAGEIN=YES.  
**Action:** Page-fix the storage containing the data to be recorded or modify the macro invocation to specify the PAGEIN=YES option. |

Example

Use GTRACE to record 200 bytes of user data plus 12 bytes for the trace record header. The user data is found at symbolic address AREA. Use an event identifier of 37. Use the formatting appendage named IMDUSR40 to control the formatting.

GTRACE DATA=AREA,LNG=200,ID=37,FID=X'40'

GTRACE DATA - List form

Use the list form of the GTRACE DATA macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form of the macro uses to store the parameters.

The list form of the GTRACE parameter list must reside below the bar.

Syntax

The list form of the DATA function of the GTRACE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><strong>name:</strong> Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede GTRACE.</td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTRACE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow GTRACE.</td>
</tr>
<tr>
<td>DATA=addr</td>
<td>addr: A-type address or register (2) - (12).</td>
</tr>
<tr>
<td>DATA64=addr</td>
<td></td>
</tr>
<tr>
<td>,LNG=nbr</td>
<td>nbr: Symbol, decimal number, hexadecimal number, or register (2) - (12).</td>
</tr>
<tr>
<td>,FID=fidname</td>
<td>fidname: Symbol, decimal number, hexadecimal number, or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=L</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are described under the standard form of the GTRACE DATA macro, with the following exception:

, MF=L

Specifies the list form of the GTRACE DATA macro.

### GTRACE DATA - Execute form

Use the execute form of the GTRACE DATA macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

### Syntax

The execute form of the DATA function of the GTRACE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede GTRACE.</td>
</tr>
<tr>
<td>GTRACE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow GTRACE.</td>
</tr>
<tr>
<td>DATA=addr</td>
<td>addr: RX address or register (2) - (12).</td>
</tr>
<tr>
<td>DATA64=addr</td>
<td></td>
</tr>
</tbody>
</table>
### GTRACE macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,LNG=nbr</td>
<td>\textit{nbr}: Symbol, decimal number, hexadecimal number, or register (2) - (12).</td>
</tr>
<tr>
<td>,ID=\textit{id}</td>
<td>\textit{id}: Symbol, decimal number, or hexadecimal number.</td>
</tr>
<tr>
<td>,FID=\textit{fidname}</td>
<td>\textit{fidname}: Symbol, decimal number, hexadecimal number, or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td>\textbf{Note}: If you omit the FID parameter on the execute form of GTRACE, the FID value defaults to zero. This default overlays the FID value that you specify on the list form of GTRACE. If you want the system to obtain the FID value from the remote problem-program parameter list, then you must specify the FID parameter as a null value by coding FID= without any \textit{fidname}.</td>
</tr>
<tr>
<td>,PAGEIN=NO</td>
<td>\textbf{Default}: PAGEIN=NO</td>
</tr>
<tr>
<td>,PAGEIN=YES</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,\textit{parm list addr})</td>
<td>\textit{parm list addr}: A-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are described under the standard form of the GTRACE DATA macro, with the following exception:

\textbf{,MF=(E,\textit{parm list addr})}

Specifies the execute form of the GTRACE DATA macro using a remote problem-program parameter list.
Chapter 21. HISMT — HIS multithreading service

Description
HISMT provides an interface to retrieve multithreading metrics at different granularity levels between the caller’s current and previous HISMT invocations.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state. PSW key 0.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB mode.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN.</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Locks may be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters and the save area must be addressable from the primary address space and must be in DREF/fixed storage.</td>
</tr>
</tbody>
</table>

Programming requirements
There must be at least 1 second between HISMT invocations for the same interval area.

The service can be invoked in any multithreading mode, including a multithreading mode of 1. The multithreading mode of a processor class is the number of active threads per core for a processor class. The multithreading mode for each processor class can be set via the IEAOPTxx parmlib member.

The caller must include the HISYMT macro to get a mapping of the interval area (INTVAREA) and constants for the return and reason codes.

HISMT exploiters may register as a listener for ENF 20 (System Information) and if a primary or secondary CPU speeds have changed, invoke HISMT to end the HISMT interval. The current CPU speeds are available in SI2V1PrimaryCPUSpeed and SI2V1SecondaryCPUSpeed fields of the STSI (see macro CSRSIIIDF). When the core speed changes during the HISMT interval, HISMT will return a warning reason code HisMT_kRsnWarn_ConfigChanged to inform users of questionable HISMT metrics due to the core state changing during the interval. See field HisMT_Hdr_Flags in the interval area header for what has changed.

The multithreading metrics requested through the HISMT service must be the same on each call for the same interval area (INTVAREA). The size of the interval area (INTVAREALEN) depends on the multithreading metrics requested, machine, and configuration. To allocate a sufficiently large interval area, you must adhere to the following protocol:

1. Obtaining sufficient storage:
a. As part of initialization, invoke the HISMT service with the requested metrics and pass an interval area that is the minimum interval area length, HisMT_Hdr_kLength. (See INTVAREA parameter for requirements on the first call for a new interval area)

b. The service will likely complete with a return code HisMT_kRetWarn and reason code HisMT_kRsnWarn_IntvAreaSmall.

c. Using the HisMT_Hdr_LengthRequired field in the interval area returned by the HISMT service, allocate a new interval area that is at least HisMT_Hdr_LengthRequired bytes long.

2. Making the first HISMT call with the new interval area:

a. Invoke the HISMT service with the same requested multithreading metrics passing the newly allocated interval area that is at least HisMT_Hdr_LengthRequired bytes long. (See INTVAREA parameter for requirements on the first call for a new interval area)

b. The service will likely complete with a return code HisMT_kRetOk. The first HISMT invocation for a new interval area marks the start of the first HISMT interval. The requested HISMT metrics will not be returned in the new interval area for the first call. Each metric will contain HisMT_Entry_kNoData since the system is unable to calculate the metrics on the very first call. (See Step 3 of the protocol below for instructions on getting the requested HISMT metrics in the interval area)

3. Making a subsequent HISMT call with the previous interval area:

a. Invoke the HISMT service with the same requested multithreading metrics passing the interval area from the previous invocation, unchanged.

b. The service will likely complete with a return code HisMT_kRetOk. This invocation marks the end of the current HISMT interval and the start of the next HISMT interval. The requested HISMT metrics will be returned in the interval area for the current interval (the time between the previous HISMT invocation and this HISMT invocation). The HISMT interval area has descriptors for each requested metric that contain information on how to locate and process each array of metric values. For all requested metrics, these descriptors contain the offset to the first element in the array of metric values, the number of elements in the array, and the size each value in the metric array. See macro HISYMT for more information.

For example, with multithreading, to calculate how much single thread capacity an MT=2 core of a particular processor class can deliver for the current workload over an interval for this workload, calculate:

\[
\text{interval} \times \text{Procclass Max Capacity Factor} / \text{HisMT_Entry_kMetricFactor}
\]

(Assembler programs can shift right by HisMT_Entry_kMetricShift bits for the division.)

Input register information

Before issuing the HISMT macro, the caller must ensure that the following general purpose register (GPR) contains the specified information.

| GPR Contents |
|---|---|
| 13 | The address of a 144-byte F4SA format save area in the primary address space |
Before issuing the HISMT macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the HISMT macro, the caller does not have to place any information into any access register (AR).

### Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>GPR</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR15 is not 0</td>
</tr>
<tr>
<td>1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

### Performance implications

None.

### Syntax

The HISMT macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>One or more blanks must precede HISMT.</td>
</tr>
<tr>
<td>HISMT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>One or more blanks must follow HISMT.</td>
</tr>
<tr>
<td>INTVAREA=xintvarea</td>
<td></td>
</tr>
</tbody>
</table>
### HISMT macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_INTVAREALEN=xintvarealen</td>
<td></td>
</tr>
<tr>
<td>_AVGTDCLASS=NO</td>
<td>Default: AVGTDCLASS=NO</td>
</tr>
<tr>
<td>_AVGTDCLASS=YES</td>
<td></td>
</tr>
<tr>
<td>_CAPCLASS=NO</td>
<td>Default: CAPCLASS=NO</td>
</tr>
<tr>
<td>_CAPCLASS=YES</td>
<td></td>
</tr>
<tr>
<td>_COREBUSYTIME=NO</td>
<td>Default: COREBUSYTIME=NO</td>
</tr>
<tr>
<td>_COREBUSYTIME=YES</td>
<td></td>
</tr>
<tr>
<td>_MAXCAPCLASS=NO</td>
<td>Default: MAXCAPCLASS=NO</td>
</tr>
<tr>
<td>_MAXCAPCLASS=YES</td>
<td></td>
</tr>
<tr>
<td>_PRODCLASS=NO</td>
<td>Default: PRODCLASS=NO</td>
</tr>
<tr>
<td>_PRODCLASS=YES</td>
<td></td>
</tr>
<tr>
<td>_PRODCORE=NO</td>
<td>Default: PRODCORE=NO</td>
</tr>
<tr>
<td>_PRODCORE=YES</td>
<td></td>
</tr>
<tr>
<td>_RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12) or (15), (GPR15),</td>
</tr>
<tr>
<td>_RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (0) or (2) - (12), (00), (GPR)</td>
</tr>
<tr>
<td>_PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>_PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>_PLISTVER=0</td>
<td></td>
</tr>
<tr>
<td>_MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>_MF=(L,list addr)</td>
<td>list addr: RS-type address or register (1) - (12)</td>
</tr>
<tr>
<td>_MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>_MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>_MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>_MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:
HISMT macro

name
An optional symbol, starting in column 1, that is the name on the HISMT macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

INTVAREA=xintvarea
is the name (RS-type), or address in register (2) - (12), of a required character input/output that contains the interval area. It must be in DREF or fixed storage. Note that the interval area must start and end on a double word boundary. The size of the interval area depends on the MT metrics requested, machine, and configuration. If this is the first HISMT request for a new interval area, the first byte of the interval area must contain binary zeroes. If this is a subsequent HISMT request, pass the interval area that was returned by the previous HISMT invocation, unchanged. Macro HISYMT contains the mapping of the interval area. The minimum amount of storage required for the request is HisMT_Hdr_kLength. See “Programming requirements” on page 229 for the protocol for passing an interval area that is long enough to accommodate the request.

INTVAREALEN=xintvarealen
is the name (RS-type), or address in register (2) - (12), of a required fullword input that contains the length of the provided interval area. The minimum INTVAREALEN required for the request is HisMT_Hdr_kLength. Note that the interval area length must be a multiple of 8 bytes.

AVGTDCLASS=NO
AVGTDCLASS=YES
Indicates whether Average Thread Density must be returned by processor class granularity and can be located using HisMT_Hdr_AvgTDClass_Desc. Average Thread Density is the average number of active threads for active cores (dispatched to physical hardware) within a processor class. If the system cannot calculate this value, the value will contain HisMt_Entry_kNoData. For example, the AVGTDCLASS will contain HisMt_Entry_kNoData when there are no cores defined in a processor class, when all cores are offline in a processor class, when all cores in a processor class change ONLINE/OFFLINE status during an HISMT interval or when no cores in a processor class were dispatched to physical hardware.
The default is AVGTDCLASS=NO.

AVGTDCLASS=NO
Average thread density is not needed for any processor class.

AVGTDCLASS=YES
Average thread density is needed for each processor class.

CAPCLASS=NO
CAPCLASS=YES
Indicates whether multithreading capacity factor metrics must be returned by processor class granularity and can be located using HisMT_Hdr_CapClass_Desc. Each metric is calculated for the current multithreading mode while the cores are dispatched to physical hardware. If the system cannot calculate this metric, the metric will contain HisMt_Entry_kNoData. For example, the CAPCLASS metric will contain HisMt_Entry_kNoData when there are no cores defined in a processor class.
The multithreading capacity factor is a metric that represents a ratio of how much work was accomplished at the current multithreading mode to the amount of work (for the same workload) that could have been accomplished while running with a multithreading mode of 1. For a multithreading mode of
1, a processor class will achieve a capacity factor ratio of 1.0 (100%) because whenever cores are dispatched to physical hardware, they are executing as much work as possible. For example, if the multithreading mode was greater than 1 (100%) and the MT capacity factor was 1.3 (130%), it means for the workload running, the cores were able to accomplish 1.3 times (130%) the work than the processor class running with a multithreading mode of 1 would have accomplished for the same workload.

The default is CAPCLASS=NO

, CAPCLASS=NO
   Multithreading capacity factor metric is not needed for any processor class.

, CAPCLASS=YES
   Multithreading capacity factor metric is needed for each processor class.

, COREBUSYTIME=NO

, COREBUSYTIME=YES
   Indicates whether core busy time must be returned by core granularity and can be located using HisMT_Hdr_CoreBusyTime_Desc.

   Core busy time is the amount of time (in milliseconds) a logical core was dispatched to a physical core over some interval. If the system cannot calculate this value, the value will contain HisMT_Entry_kNoData. For example, the COREBUSYTIME will contain HisMt_Entry_kNoData for an undefined core.

The default is COREBUSYTIME=NO

, COREBUSYTIME=NO
   Core busy time is not needed for any core.

, COREBUSYTIME=YES
   Core busy time is needed for each core.

, MAXCAPCLASS=NO

, MAXCAPCLASS=YES
   Indicates whether multithreading maximum capacity factor metrics must be returned by processor class granularity and can be located using HisMT_Hdr_MaxCapClass_Desc. Each metric is calculated for the current multithreading mode while the cores are dispatched to physical hardware. If the system cannot calculate this metric, the metric will contain HisMT_Entry_kNoData. For example, the MAXCAPCLASS metric will contain HisMt_Entry_kNoData when there are no cores defined in a processor class.

   Multithreading max capacity factor is a metric that represents a ratio of the maximum amount of work that can be accomplished using all active threads at the current multithreading mode to the amount of work (for the same workload) that would have been accomplished while running with a multithreading mode of 1. For a multithreading mode of 1, a processor class will achieve a max capacity factor ratio of 1.0 (100%) because whenever cores are dispatched to physical hardware, they are executing the maximum amount of work as possible. For example, if the multithreading mode is greater than 1 and the multithreading max capacity factor was 1.4 (140%), it means that if the processor class was able to achieve a productivity ratio of 1.0, then the workload running would be able to accomplish 1.4 times (140%) the work a processor class running with a multithreading mode of 1 would have accomplished for the same workload.

The default is MAXCAPCLASS=NO
Multithreading max capacity factor metric is not needed for any processor class.

Multithreading max capacity factor metric is needed for each processor class.

Indicates whether multithreading productivity metrics must be returned by processor class granularity and can be located using HisMT_Hdr_ProdClass_Desc. Each metric is calculated for the current multithreading mode while the cores are dispatched to physical hardware. If the system cannot calculate this metric, the metric will contain HisMT_Entry_kNoData. For example, the PRODCLASS metric will contain HisMt_Entry_kNoData when there are no cores defined in a processor class.

Multithreading productivity is a metric that represents a ratio of how much work was accomplished to the maximum amount of work that could have been accomplished. For a multithreading mode of 1, a processor class will achieve a productivity ratio of 1.0 (100%) because whenever cores are dispatched to physical hardware, they are executing as much work as possible.

For example, if the multithreading mode is greater than 1 and the productivity ratio is 0.93 (93%), it means the active threads on all cores accomplished 93% of the work that could have been accomplished while dispatched to physical hardware. Typically, when the multithreading productivity is less than 1.0 (<100%), it is because there were times when the cores were dispatched to physical hardware and one or more threads on those cores were in a wait state because they had no work to run.

The default is PRODCLASS=NO

Multithreading productivity metric is not needed for any processor class.

Multithreading productivity metric is needed for each processor class.

Indicates whether multithreading productivity metrics must be returned by core granularity and can be located using HisMT_Hdr_ProdCore_Desc. Each metric is calculated for the current multithreading mode while the cores are dispatched to physical hardware. If the system cannot calculate this metric, the metric will contain HisMT_Entry_kNoData. For example, the PRODCORE metric will contain HisMt_Entry_kNoData for an undefined or offline core.

See the PRODCLASS keyword for information about multithreading productivity.

The default is PRODCORE=NO

Multithreading productivity metric is not needed for any core.

Multithreading productivity metric is needed for each core.

An optional output parameter into which the return code is to be copied from...
GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) or (15), (GPR15), (REG15), or (R15).

\[RSNCODE=rsncode\]

An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2) - (12), (00), (GPR0), (GPR00), (REG0), (REG00), or (R0).

\[PLISTVER=IMPLIED_VERSION\], \[PLISTVER=MAX\], \[PLISTVER=0\]

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. PLISTVER is the only key allowed on the list form of MF and determines which parameter list is generated. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

\[MF=S\], \[MF=(L,list addr)\], \[MF=(L,list addr,attr)\], \[MF=(L,list addr,0D)\], \[MF=(E,list addr)\], \[MF=(E,list addr,COMPLETE)\]

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The
list form defines an area of storage that the execute form uses to store the
parameters. Only the PLISTVER parameter may be coded with the list form of
the macro.

Use MF=E to specify the execute form of the macro. Use the execute form
together with the list form of the macro for applications that require reentrant
code. The execute form of the macro stores the parameters into the storage area
defined by the list form, and generates the macro invocation to transfer control
to the service.

\list addr

The name of a storage area to contain the parameters. For MF=S and
MF=E, this can be an RS-type address or an address in register (1) - (12).

\attr

An optional 1- to 60-character input string that you use to force boundary
alignment of the parameter list. Use a value of 0F to force the parameter
list to a word boundary, or 0D to force the parameter list to a doubleword
boundary. If you do not code attr, the system provides a value of 00.

\complete

Specifies that the system is to check for required parameters and supply
defaults for omitted optional parameters.

**ABEND codes**

None.

**Return and reason codes**

When the HISMT macro returns control to your program:

- GPR 15 (and retcode, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncole, when you code
RSNCODE) contains a reason code.

Macro HISYSERV provides equate symbols for the return and reason codes. Note
carefully that bits 0 - 15 of the reason code may contain component diagnostic data
and must not be assumed to be 0.

The following table identifies the hexadecimal return and reason codes and the
equate symbol associated with each reason code. IBM support personnel may
request the entire reason code, including the xxxx value.

**Table 26. Return and reason codes for the HISMT macro**

<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason code</th>
<th>Equate symbol, meaning, and action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>Equate Symbol: HisMT_kRetOk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: HISMT request successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Processing continues.</td>
</tr>
<tr>
<td>4</td>
<td>–</td>
<td>Equate Symbol: HisMT_kRetWarn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: Warning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Refer to the action provided with the specific reason code.</td>
</tr>
</tbody>
</table>
### Table 26. Return and reason codes for the HISMT macro (continued)

<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason code</th>
<th>Equate symbol, meaning, and action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>xxxx0401</td>
<td><strong>Equate Symbol</strong>: HisMT_kRsnWarn_IntvAreaSmall</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: The interval area provided was large enough to hold the minimum amount of data required for the request, but not large enough to hold all of the data requested.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Obtain a larger interval area using the HisMT_Hdr_LengthRequired field returned in the request's INTVAREA. Then call the service with the newly allocated interval area. (See “Programming requirements” on page 229 for the protocol for passing a new interval area that is at least HisMT_Hdr_LengthRequired bytes long)</td>
</tr>
<tr>
<td>4</td>
<td>xxxx0402</td>
<td><strong>Equate Symbol</strong>: HisMT_kRsnWarn_ConfigChanged</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: The system configuration was changed during the interval between the last HISMT call and the current HISMT call. The metric values returned in the interval area is questionable since the configuration was not consistent during the interval. See field HisMT_Hdr_FLAGS in the interval area header for what has changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: The metric values returned from this call can be ignored.</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
<td><strong>Equate Symbol</strong>: HisMT_kRetUser</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: HISMT request failed due to a user error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Refer to the action provided with the specific reason code</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0801</td>
<td><strong>Equate Symbol</strong>: HisMT_kRsnUser_InvVersion</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: The version for the parameter list specified is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Check for possible storage overlay.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0802</td>
<td><strong>Equate Symbol</strong>: HisMT_kRsnUser_InconsistentIntvArea</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: The HISMT call for this interval does not match the previous HISMT call for the provided interval area.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: If this is the first invocation for this interval area, follow the protocol described in “Programming requirements” on page 229 to provide an interval area. If this is a subsequent call to HISMT for the interval area, make sure all subsequent calls to HISMT with that interval area request the same MT metrics.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0803</td>
<td><strong>Equate Symbol</strong>: HisMT_kRsnUser_IntvLenTooSmall</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: The interval area is less than HisMT_Hdr_kLength bytes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Ensure the interval area length and the storage provided for the interval area is at least HisMT_Hdr_kLength bytes long. (See “Programming requirements” on page 229 for the protocol for obtaining big enough storage to contain the requested MT metrics)</td>
</tr>
</tbody>
</table>
Table 26. Return and reason codes for the HISMT macro (continued)

<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason code</th>
<th>Equate symbol, meaning, and action</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>xxx0804</td>
<td><strong>Equate Symbol:</strong> HisMT_kRsnUser_UnknownDataInIntvArea</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The interval area for this HISMT call contains some unexpected data. A storage overlay may have occurred.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Issue HISMT request with a new interval area.</td>
</tr>
<tr>
<td>8</td>
<td>xxx0805</td>
<td><strong>Equate Symbol:</strong> HisMT_kRsnUser_IntvAreaNotAligned</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The interval area provided for this HISMT call is not on a doubleword boundary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Make sure that the interval area is on a doubleword boundary.</td>
</tr>
<tr>
<td>8</td>
<td>xxx0806</td>
<td><strong>Equate Symbol:</strong> HisMT_kRsnUser_IntvLenNot8ByteMultiple</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The interval area length provided for this HISMT call is not a multiple of 8-bytes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Provide an interval area whose length is an 8-byte multiple.</td>
</tr>
<tr>
<td>8</td>
<td>xxx0807</td>
<td><strong>Equate Symbol:</strong> HisMT_kRsnUser_UnknownEyeCatcher</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The eye catcher in the interval area for this HISMT call is unexpected. A storage overlay may have occurred.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Issue a HISMT request with a new interval area. The first byte must contain binary zeroes.</td>
</tr>
<tr>
<td>10</td>
<td>–</td>
<td><strong>Equate Symbol:</strong> HisMT_kRetUnknown</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> Unexpected failure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Refer to the action provided with the specific reason code.</td>
</tr>
<tr>
<td>10</td>
<td>xxxx1001</td>
<td><strong>Equate Symbol:</strong> HisMT_kRsnUnknown_Unknown</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> Unexpected failure. The state of the request is unpredictable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Contact your system programmer.</td>
</tr>
</tbody>
</table>

Example

Operation

Requesting CAPCLASS, MAXCAPCLASS, PRODCLASS and PRODCORE metrics with HISMT:

1. Invoke HISMT with the requested metrics and pass an interval area that is the minimum interval area length, HisMT\_Hdr\_kLength.
2. Use the HisMT\_Hdr\_LengthRequired field in the interval area returned by the HISMT service and, allocate a new interval area that is at least HisMT\_Hdr\_LengthRequired bytes long.
3. Invoke HISMT for the first time with the requested metrics and pass the new interval area with sufficient storage.
4. Invoke HISMT for a subsequent call with the same requested metrics and pass the interval area from the previous invocation, unchanged. The requested HISMT metrics will be returned in the interval area for the current interval.
The code is as follows:

********************************************************************
* Invoke the HISMT service with the requested metrics and pass *
* an interval area that is the minimum interval area length, *
* HisMT_Hdr_kLength.                                     *
* Note:                                                   *
* On the first call, the first byte of the interval area    *
* (HisMT_Hdr_EyeCatcherFirstChar) must contain binary zeroes *
********************************************************************

LA R8, LHisMT_Header
Using HisMTHdr, R8
MVI HisMT_Hdr_EyeCatcherFirstChar, X‘00’
HISMT MF=(E, serviceList), INTVAREA=LHisMT_Header, X
INTVAREALEN=HisMT_Hdr_kLength, X
PRODCLASS=YES, PRODCORE=YES, X
MAXCAPCLASS=YES, CAPCLASS=YES, X
RETCODE=LRetCode, RSNCODE=LRsnCode

********************************************************************
* Check return and reason code.                             *
* If HISMT requests a larger area, save required length.     *
* Obtain new storage with required length (code not shown)   *
********************************************************************

LHI R2, HisMT_kRetWarn
L R3, LRetCode
CLR R2, R3
JNE INTVAREA_UNEXP
LHI R4, HisMT_kRsnWarn_IntvAreaSmall
L R5, LRsnCode
CLR R4, R5
JE INTVAREA_WARN

INTVAREA_UNEXP DS 0H
* Place code to handle unexpected return/reason codes here *

INTVAREA_WARN DS 0H
L R7, HisMT_Hdr_LengthRequired
* Place code to obtain storage of length HISMT_HDR_LENGTHREQUIRED and
* save address in R8
*
********************************************************************
* Invoke the HISMT service with the requested metrics and pass *
* an interval area that is the required interval area length, *
* HISMT_HDR_LENGTHREQUIRED. Save required interval area length *
* in LHisMT_INTVAREA_LEN
* Note:                                                   *
* On the first call, the first byte of the interval area    *
* (HisMT_Hdr_EyeCatcherFirstChar) must contain binary zeroes *
********************************************************************

ST R7, LHisMT_INTVAREA_LEN
Using HisMT_Hdr, R8
MVI HisMT_Hdr_EyeCatcherFirstChar, X‘00’

EVN_LOOP DS 0H
HISMT MF=(E, serviceList), INTVAREA=HisMT_Hdr, X
INTVAREALEN=LHisMT_INTVAREA_LEN, X
PRODCLASS=YES, PRODCORE=YES, X
MAXCAPCLASS=YES, CAPCLASS=YES, X
RETCODE=LRetCode, RSNCODE=LRsnCode

* Place code to check return/reason codes here
On a subsequent HISMT call, invoke the HISMT service with the same requested MT metrics passing the interval area from the previous invocation, unchanged.

For a subsequent HISMT call, place code to process the returned metrics for the current interval here.

J EVN_LOOP

DynArea DSECT
LRetCode DS F
LRsnCode DS F
LHisMT_Header DS CL(HisMT_Hdr_kLength)
LHisMT_INTVAREA_LEN DS F
   HISMT MF=(L,serviceList)
   POP USING
   HISYMT
Chapter 22. HISSERV macro — HISSERV Service

Description

HISSERV provides an interface to begin profiling and retrieve instrumentation data from the system. There are currently two types of instrumentation data:

Events

Events are recorded at the CPU or core level. As events occur, they are captured and recorded, to be queried at any interval determined by software. Events are grouped into event types, which can be enabled and disabled independently of each other.

Sampling

At predetermined intervals, a sample representing the current state of a CPU is stored into a Sampling Data Buffer (SDBs). As SDBs are filled software is notified allowing the software to process the full SDBs. The SDBs are then cleared to be reused by the hardware. There are different sampling types that can be enabled and disabled independently of each other, however a profiler can only indicate its intention to receive sampling data. The sampling frequency, as well as which sampling types are enabled are determined by the service parameters specified on a F hisproc,SERVICE command.

Specifically, with HISSERV you can do the following:

- Query for event info such as determining which events and event types are available. (REQUEST=QUERY,TYPE=EVENT).
- Query for sampling info such as the sampling interval and which sampling types are available. (REQUEST=QUERY,TYPE=SAMPLE).
- Query for statistics of whomever is currently profiling the system. (REQUEST=QUERY,TYPE=PROFILERS).
- Begin profiling the system, indicating to the system the intention of collecting one or more event types and/or sampling data. Requests that require a PROFILETKN must first use this to identify itself as wanting to profile the system. (REQUEST=PROFILE,ACTION=START)
- Stop profiling the system. The PROFILETKN is no longer useable and when the last profiler stops profiling the system, any unnecessary resources are released. (REQUEST=PROFILE,ACTION=STOP)
- Query for event data provided by the service. Requires a PROFILETKN. (REQUEST=QUERY,TYPE=EVENTDATA).

The HISSERV service is only enabled when the HIS address space has been initialized. The dynamic exit HIS.SERVSTAT can be used to be notified when the service has been enabled or disabled.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PKM 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
</tbody>
</table>
Environmental factor
AMODE: 31- or 64-bit
ASC mode: Primary or access register (AR)
Interrupt status: Enabled for I/O and external interrupts
Locks: The caller must not be holding any locks.
Control parameters: Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

The user-provided answer area (via the ANSAREA parameter) has the same requirements and restrictions as the control parameters.

The user-provided CPU or core bitmask (via the CPUMASK parameter) has the same requirements and restrictions as the control parameters.

Programming Requirements
The caller should include the HISYSERV macro to get equate symbols for the return and reason codes.

The caller must include the HISYSERV macro to get a mapping of the output area provided via the ANSAREA parameter for REQUEST=QUERY.

The caller must include the HISYEXIT macro to get a mapping of the parameter area passed to the exit routine specified by the EXITRTN parameter for REQUEST=PROFILE,ACTION=START requests.

The caller must include the HISYSMPX macro to get a mapping of the parameter area passed to the exit routine specified by the EXITRTN parameter for REQUEST=PROFILE,ACTION=START requests, when the profiler requests sampling data (SAMPLE=YES).

Restrictions
The caller must not have functional recovery routines (FRRs) established.

Input Register Information
Before issuing the HISSERV macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the HISSERV macro, the caller does not have to place any information into any access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR15 is not 0</td>
</tr>
<tr>
<td>1</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
2-13  Unchanged
14  Used as a work register by the system
15  Return code

When control returns to the caller, the ARs contain:

**Register**

**Contents**

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance Implications**

None.

**Syntax**

The HISSERV macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
</tbody>
</table>

One or more blanks must precede HISSERV.

| HISSERV | |
|---------| |

One or more blanks must follow HISSERV.

| REQUEST=PROFILE |
| REQUEST=QUERY |

| ACTION=START |
| ACTION=STOP |

| ,OUTPROFILETKN=OUTPROFILETKN | outprofiletkn: RS-type address or address in register (2) - (12) |
| ,EVENT=event | event: RS-type address or address in register (2) - (12) |
| ,EVENT=NO_EVENT | Default: EVENT=NO_EVENT |
| ,SAMPLE=NO | Default: SAMPLE=NO |
### HISSERV macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_SAMPLE= YES</code></td>
<td></td>
</tr>
<tr>
<td><code>_NAME=name</code></td>
<td><em>name</em>: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td><code>_EXITRTN=exitrtn</code></td>
<td><em>exitrtn</em>: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td><code>_PROFILETKN=profiletkn</code></td>
<td><em>profiletkn</em>: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td><code>_ANSAREA=ansarea</code></td>
<td><em>ansarea</em>: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td><code>_ANSLEN=anslen</code></td>
<td><em>anslen</em>: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td><code>_TYPE=EVENTDATA</code></td>
<td></td>
</tr>
<tr>
<td><code>_TYPE=EVENT</code></td>
<td></td>
</tr>
<tr>
<td><code>_TYPE=SAMPLE</code></td>
<td></td>
</tr>
<tr>
<td><code>_TYPE=PROFILERS</code></td>
<td></td>
</tr>
<tr>
<td><code>_PROFILETKN=profiletkn</code></td>
<td><em>profiletkn</em>: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td><code>_CPUMASK=cpumask</code></td>
<td><em>cpumask</em>: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td><code>_CPUMASK= ALL</code></td>
<td>Default: CPUMASK=ALL</td>
</tr>
<tr>
<td><code>_RETCODE=retcode</code></td>
<td><em>retcode</em>: RS-type address or register (2) - (12) or (15), (GPR15),</td>
</tr>
<tr>
<td><code>_RSNCODE=rsncode</code></td>
<td><em>rsncode</em>: RS-type address or register (0) or (2) - (12), (00), (GPR)</td>
</tr>
<tr>
<td><code>_PLISTVER=IMPLIED_VERSION</code></td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td><code>_PLISTVER= MAX</code></td>
<td></td>
</tr>
<tr>
<td><code>_PLISTVER= 0</code></td>
<td></td>
</tr>
<tr>
<td><code>_MF=S</code></td>
<td>Default: MF=S</td>
</tr>
<tr>
<td><code>_MF=(L,list addr)</code></td>
<td><em>list addr</em>: RS-type address or register (1) - (12)</td>
</tr>
<tr>
<td><code>_MF=(L,list addr,attr)</code></td>
<td></td>
</tr>
<tr>
<td><code>_MF=(L,list addr,0D)</code></td>
<td></td>
</tr>
<tr>
<td><code>_MF=(E,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>_MF=(E,list addr,COMPLETE)</code></td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:
HISSERV macro

name
An optional symbol, starting in column 1, that is the name on the HISSERV macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=PROFILE
REQUEST=QUERY
A required parameter, used to indicate the type of request.

REQUEST=PROFILE
indicates the intention to start or stop profiling the system.

REQUEST=QUERY
indicates to query the service.

,ACTION=START
,ACTION=STOP
When REQUEST=PROFILE is specified, a required parameter, used to indicate the profiling action to take. Note that a PROFILE action cannot be requested from within a profiler’s exit routine, nor should it be requested from a work unit holding a resource required by a profiler’s exit routine.

,ACTION=START
indicates to start profiling the system.

,ACTION=STOP
indicates to stop profiling the system. When profiling for sampling data, the exit routine specified by EXITRTN will receive a final sampling related callback, with the HisSmpParmFlgs_Last flag on, for each CPU that is currently sampling. Note this service call will not return until after the EXITRTN has handled every CPU’s final sampling callback. If no more profilers are profiling the system, all resources associated with profiling will be released.

,OUTPROFILETKN=outprofiletkn
When ACTION=START and REQUEST=PROFILE are specified, a required output parameter, into which the unique profiler token to identify this profiler will be returned. If this is the first profiler of the system, resources associated with profiling will be obtained and held until the last profiler stops profiling the system.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,EVENT=event
,EVENT=NO_EVENT
When ACTION=START and REQUEST=PROFILE are specified, an optional input parameter, which should contain the event types to profile. When querying the event data, only the event types specified when starting to PROFILE the system will be returned. If the event types are not authorized at the time of the PROFILE request the request will be remembered. Later if the system becomes authorized for that event type, it will be returned in any subsequent event data query. The storage is mapped by HisEvnTyp in macro HISYSERV and must be a subset of the event type data returned in HisEvn_ValidEvnTyp, which is returned in the REQUEST=QUERY,TYPE=EVENT request. If EVENT=NO_EVENT or the storage passed in is binary zeroes, the profiler will not be able to query for event data. The default is NO_EVENT.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32 bit field.
When ACTION=START and REQUEST=PROFILE are specified, an optional parameter, which is used to determine whether to receive callbacks with sampling data. When SAMPLE=YES, the exit routine defined in the EXITRTN parameter will be called as Sampling Data Blocks (SDBs) become available. If sampling is not authorized at the time of the PROFILE request, the request will be remembered. Later, if the system becomes authorized for sampling, it will begin providing sampling data. The types of sampling entries returned by the service are dependent on the configuration of the service by the SAMPTYPE parameter from the most recent F HIS,SERVICE or F HIS,BEGIN command. The default is SAMPLE=NO.

\_SAMPLE=NO

Do not receive callbacks with sampling data.

\_SAMPLE=YES

Receive callbacks with sampling data.

\_NAME=\_name

When ACTION=START and REQUEST=PROFILE are specified, a required input parameter, which should contain the unique name identifying this profiler. The name should use EBCDIC characters from among the set of alphanumerics. The NAME will be returned in any QUERY,TYPE=PROFILE queries to identify this profiler. This name will also be displayed as output from the D HIS command. The NAME specified should be one that easily identifies the product requesting the profiling, for example, the HIS supplied profiler starts with "HIS". The service will not allow another name starting with "HIS". The NAME must be unique for each profiler registered with the service.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

\_EXITRTN=\_exitrtn

When ACTION=START and REQUEST=PROFILE are specified, a required input parameter, which should contain the name of the exit routine that will be called when the service needs to notify the profiler for some reason, such as for sampling callbacks. The exit routine must reside in LPA, the LNKLIST LNKLIST concatenation, or the nucleus. The interface to the exit routine is described in macro HISYEXIT. The EXITRTN must be unique for each profiler registered with the service.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

\_PROFILETKN=\_profiletkn

When ACTION=STOP and REQUEST=PROFILE are specified, a required input parameter, which should contain the profiler's unique token received from this profiler's REQUEST=PROFILE,ACTION=START request (parameter OUTPROFILETKN.) The profiler's token will no longer be usable.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

\_ANSAREA=\_ansarea

When REQUEST=QUERY is specified, a required input parameter, which will be used by the service to store information associated with the query request. Macro HISYSERV contains mappings of the answer areas to provide, the size...
of the area depends on the type of query being requested. The minimum amount of storage required for the request to be successful and return a subset of the data, is HisAns_kLength.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ANSLEN=anslen
When REQUEST=QUERY is specified, a required input parameter, which should contain the length of the provided answer area. The length depends on the query that is requested. The minimum ANSLEN required for the request to be successful and return a subset of the data, is HisAns_kLength.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field, or specify a literal decimal value.

,TYPE=EVENTDATA
,TYPE=EVENT
,TYPE=SAMPLE
,TYPE=PROFILERS
When REQUEST=QUERY is specified, a required parameter, used to indicate the type of query to process.

,TYPE=EVENTDATA
indicates to process a query event data request. ANSAREA should point to storage that will be mapped by HisEvnData in macro HISYSERV. It is possible for the amount of storage required to change between two consecutive TYPE=EVENTDATA queries, depending on which event types are currently authorized in the system and which CPUs or cores are currently online.

,TYPE=EVENT
indicates to process a query event request. ANSAREA should point to storage that will be mapped by HisEvn in macro HISYSERV.

,TYPE=SAMPLE
indicates to process a query sample request. ANSAREA should point to storage that will be mapped by HisSmp in macro HISYSERV.

,TYPE=PROFILERS
indicates to process a query profiler info request. ANSAREA should point to storage that will be mapped by HisProf in macro HISYSERV. It is possible for the amount of storage required to change between two consecutive TYPE=PROFILERS queries, depending on the current number of profilers in the system.

,PROFILETKN=profiletkn
When TYPE=EVENTDATA and REQUEST=QUERY are specified, a required input parameter, which should contain the profiler’s unique token received from this profiler’s REQUEST=PROFILE, ACTION=START request (parameter OUTPROFILETKN).

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,CPUMASK=cpumask
,CPUMASK=ALL
When TYPE=EVENTDATA and REQUEST=QUERY are specified, an optional input parameter representing a bitmask of which CPUs and/or cores to query event data. The bitmask should be ECVTMaxMPNumBytesInMask bytes long. Bit 0 represents CPU 0’s event data as well as core 0’s event data to query, and
so forth up to the bit position at CVTMAXMP. If requesting all CPUs and all
cores specify ALL, pass in a CPUMASK of binary ones, or omit the CPUMASK
parameter. The default is ALL.

To code: Specify the RS-type address, or address in register (2)-(12), of a
character field.

`RETCODE=retcode`
An optional output parameter into which the return code is to be copied from
GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without
parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or
(15), (GPR15), (REG15), or (R15).

`RSNCODE=rsncode`
An optional output parameter into which the reason code is to be copied from
GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or
without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or
(2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

`PLISTVER=IMPLIED_VERSION`
`PLISTVER=MAX`
`PLISTVER=0`
An optional input parameter that specifies the version of the macro. PLISTVER
determines which parameter list the system generates. PLISTVER is an
optional input parameter on all forms of the macro, including the list form.
When using PLISTVER, specify it on all macro forms used for a request and
with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters
  specified on the request to be processed. If you omit the PLISTVER
  parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible.
  This size might grow from release to release and affect the amount of
  storage that your program needs.

If you can tolerate the size change, IBM recommends that you always
specify PLISTVER=MAX on the list form of the macro. Specifying MAX
ensures that the list-form parameter list is always long enough to hold all
the parameters you might specify on the execute form, when both are
assembled with the same level of the system. In this way, MAX ensures that
the parameter list does not overwrite nearby storage.

- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- **IMPLIED_VERSION**
- **MAX**
- A decimal value of 0

`MF=S`
`MF=(L,list addr)`
`MF=(L,list addr,attr)`
`MF=(L,list addr,0D)`
`MF=(E,list addr)`
`MF=(E,list addr,COMPLETE)`
An optional input parameter that specifies the macro form.
Use MF=S to specify the standard form of the macro, which builds an inline
parameter list and generates the macro invocation to transfer control to the
service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with
the execute form of the macro for applications that require reentrant code. The
list form defines an area of storage that the execute form uses to store the
parameters. Only the PLISTVER parameter may be coded with the list form of
the macro.

Use MF=E to specify the execute form of the macro. Use the execute form
together with the list form of the macro for applications that require reentrant
code. The execute form of the macro stores the parameters into the storage area
defined by the list form, and generates the macro invocation to transfer control
to the service.

\texttt{,list addr}

The name of a storage area to contain the parameters. For MF=S and
MF=E, this can be an RS-type address or an address in register (1)-(12).

\texttt{,attr}

An optional 1- to 60-character input string that you use to force boundary
alignment of the parameter list. Use a value of 0F to force the parameter
list to a word boundary, or 0D to force the parameter list to a doubleword
boundary. If you do not code \texttt{attr}, the system provides a value of 0D.

\texttt{,COMPLETE}

Specifies that the system is to check for required parameters and supply
defaults for omitted optional parameters.

**ABEND Codes**

None.

**Return and Reason Codes**

When the HISSERV macro returns control to your program:

- GPR 15 (and \texttt{retcode}, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and \texttt{rsncode}, when you code
  RSNCODE) contains a reason code.

Macro HISYSERV provides equate symbols for the return and reason codes. Note
carefully that bits 0-15 of the reason code may contain component-diagnostic data
and must not be assumed to be 0.

The following table identifies the hexadecimal return and reason codes and the
equate symbol associated with each reason code. IBM support personnel may
request the entire reason code, including the xxxx value.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>—</td>
<td>\texttt{Equate Symbol: Hiserv_kRetOk}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{Meaning: HISSERV request successful.}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{Action: Processing continues.}</td>
</tr>
<tr>
<td>Return Code</td>
<td>Reason Code</td>
<td>Equate Symbol Meaning and Action</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>4</td>
<td>—</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRetWarn</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: Warning</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Refer to the action provided with the specific reason code.</td>
</tr>
<tr>
<td>4</td>
<td>xxxx0401</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnWarn_AnsAreaSmall</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: For REQUEST=QUERY, the answer area provided was large enough to hold the minimum amount of data required for the request, but not large enough to hold all of the data requested.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Obtain a larger answer area using the HisAns_LengthRequire field returned in the request's ANSAREA.</td>
</tr>
<tr>
<td>8</td>
<td>—</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRetUser</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: HISSERV request failed due to a user error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Refer to the action provided with the specific reason code.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0801</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnUser_BadParmArea</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: Unable to access parameter area.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Check for possible storage overlay.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0802</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnUser_BadParmAreaALET</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: Bad parameter area ALET.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Make sure that the ALET associated with the parameter area is valid. The access register might not have been set up correctly.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0803</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnUser_BadVersion</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: Bad version for the parameter list was specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Check for possible storage overlay.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0804</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnUser_SrbMode</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: This function is only available in task mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Use function in task mode.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0805</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnUser_NotEnabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: This function is only available to enabled programs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Use function while enabled.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0806</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnUser_LocksHeld</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: This function is only available to unlocked programs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Use function while unlocked.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0807</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnUser_CallerFRR</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: This function is only available to programs that have not established an FRR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Retry the request without an FRR established.</td>
</tr>
</tbody>
</table>
### Table 27. Return and Reason Codes for the HISSERV Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>xxxx0808</td>
<td>Hisserv_kRsnUser_BadRequest</td>
<td>A Bad request was made to the service.</td>
<td>Check for possible storage overlay.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0809</td>
<td>Hisserv_kRsnUser_BadProfTkn</td>
<td>Token specified was not a valid token.</td>
<td>Use a valid token provided by the REQUEST=PROFILE,ACTION=START request.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx080A</td>
<td>Hisserv_kRsnUser_NameInUse</td>
<td>The name requested is already in use.</td>
<td>Provide a NAME that is unique to the service.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx080B</td>
<td>Hisserv_kRsnUser_InvName</td>
<td>The name requested is invalid.</td>
<td>Provide a valid NAME, it cannot begin with HIS.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx080C</td>
<td>Hisserv_kRsnUser_ExitRtnNotFound</td>
<td>The exit routine specified wasn't found.</td>
<td>Ensure the exit routine specified exists in LPA, the LNKLS concatenation, or the nucleus.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx080D</td>
<td>Hisserv_kRsnUser_ExitRtnInUse</td>
<td>The exit routine specified is already in use.</td>
<td>A different exit routine must be provided.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx080E</td>
<td>Hisserv_kRsnUser_BadEvnTyp</td>
<td>For REQUEST=PROFILE,ACTION=START requests, one or more event types specified could not be properly configured for because it is not allowed. Only event types returned in the HisEvn_ValidEvnTyp field of a REQUEST=QUERY,TYPE=EVENT request can be requested.</td>
<td>Ensure the event types being requested are a subset of the event types returned in the HisEvn_ValidEvnTyp field of a REQUEST=QUERY,TYPE=EVENT request.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx080F</td>
<td>Hisserv_kRsnUser_BadProfReq</td>
<td>A bad PROFILE request was made to the service.</td>
<td>Check for possible storage overlay.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0810</td>
<td>Hisserv_kRsnUser_BadProfStart</td>
<td>For REQUEST=PROFILE,ACTION=START, a bad request was made. At least one event type or sampling should be requested when starting to profile the system.</td>
<td>Request at least one event type or sampling.</td>
</tr>
</tbody>
</table>
### HISSERV macro

#### Table 27. Return and Reason Codes for the HISSERV Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>xxxx0811</td>
<td>Hisserv_kRsnUser_BadQuery</td>
<td>For REQUEST=QUERY, a bad query was requested.</td>
<td>Check for possible storage overlay.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0812</td>
<td>Hisserv_kRsnUser_BadAnsArea</td>
<td>For REQUEST=QUERY, unable to access answer area.</td>
<td>Provide a valid answer area.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0813</td>
<td>Hisserv_kRsnUser_BadAnsAreaALET</td>
<td>Bad answer area ALET.</td>
<td>Make sure that the ALET associated with the answer area is valid. The access register might not have been set up correctly.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0814</td>
<td>Hisserv_kRsnUser_AnsLenTooSmall</td>
<td>For REQUEST=QUERY, the answer area length is incorrect.</td>
<td>Ensure the answer area length and the storage provided as the answer area is at least HisAns_kLength bytes long.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0815</td>
<td>Hisserv_kRsnUser_BadCpuMask</td>
<td>For REQUEST=QUERY,TYPE=EVENTDATA requests, unable to access the CPU mask.</td>
<td>Provide a valid CPU mask.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0816</td>
<td>Hisserv_kRsnUser_BadCpuMaskALET</td>
<td>Bad CPU mask ALET.</td>
<td>Make sure that the ALET associated with the CPU mask is valid. The access register might not have been set up correctly.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0817</td>
<td>Hisserv_kRsnUser_NoEvnTyp</td>
<td>For REQUEST=QUERY,TYPE=EVENTDATA requests, the profiler making the request is not profiling events.</td>
<td>When registering with the system to profile, indicate the intention to profile events using the EVENT= parameter.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0818</td>
<td>Hisserv_kRsnUser_InvProfChange</td>
<td>A REQUEST=PROFILE request was made from a profiler’s exit routine.</td>
<td>A REQUEST=PROFILE request cannot be made from a profiler’s exit routine.</td>
</tr>
<tr>
<td>C</td>
<td>—</td>
<td>Hisserv_kRetEnv</td>
<td>Environmental error</td>
<td>Refer to the action provided with the specific reason code.</td>
</tr>
</tbody>
</table>
### Table 27. Return and Reason Codes for the HISSERV Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>xxxx0C01</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnEnv_NotAvailable</td>
<td><strong>Meaning</strong>: Function is not available.</td>
<td><strong>Action</strong>: This function is only available when the HIS address space is running.</td>
</tr>
<tr>
<td>C</td>
<td>xxxx0C02</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnEnv_NotReady</td>
<td><strong>Meaning</strong>: Function is available but is not currently ready to accept requests.</td>
<td><strong>Action</strong>: Retry the request.</td>
</tr>
<tr>
<td>C</td>
<td>xxxx0C03</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnEnv_NoStorage</td>
<td><strong>Meaning</strong>: There was not enough storage in HIS private storage to complete the request.</td>
<td><strong>Action</strong>: Contact your system programmer.</td>
</tr>
<tr>
<td>10</td>
<td>—</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRetUnk</td>
<td><strong>Meaning</strong>: Unexpected failure.</td>
<td><strong>Action</strong>: Refer to the action provided with the specific reason code.</td>
</tr>
<tr>
<td>10</td>
<td>xxxx1001</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnUnk_Unk</td>
<td><strong>Meaning</strong>: Unexpected failure. The state of the request is unpredictable.</td>
<td><strong>Action</strong>: Contact your system programmer.</td>
</tr>
<tr>
<td>10</td>
<td>xxxx1002</td>
<td><strong>Equate Symbol</strong>: Hisserv_kRsnUnk_QueryCpu</td>
<td><strong>Meaning</strong>: For REQUEST=QUERY,TYPE=EVENTDATA, while attempting to query a CPU’s event data an unknown error occurred.</td>
<td><strong>Action</strong>: Contact your system programmer.</td>
</tr>
</tbody>
</table>

### Example

**Operation**

1. Profile for all available event types, and sampling.
2. Query the event types on 1 minute intervals for an hour, ensuring all event data is returned each query.
3. Process Sampling Data Blocks (SDBs) as they become available.
4. Stop profiling the system.

The code is as follows.

```plaintext
* Query Events to determine what can be enabled. *
HISERV MF=(E,serviceList),REQUEST=QUERY, *
ANSAREA=EvnAnsArea,ANSLEN=EvnAnsLen, *
TYPE=EVENT,RETCODE=LRetCode, *
RSNCODE=LRSncode
JZ EVNAREA_GOOD
```

---

---
HISSERV macro

```
LA R4, Hisserv_kRetWarn
CLR R5, R4
JE EVNAREA_WARN
EVNAREA_BAD DS 0H

* Place code to check bad return/reason codes here
*
EVNAREA_WARN DS 0H
L R5, LRsncode
NILH R5, 0
CHI R5, Hisserv_kRsncWarn_AnsAreaSmall
JNE EVNAREA_BAD

* Place code to handle obtaining more storage for the
* ANSAREA, and repeat the request if necessary.
*
EVNAREA_GOOD DS 0H
LA R8, EvnAnsArea
Using HisEvn, R8

*******************************************************************************
* Start profiling the system. Output from the previous query is used as input to this query, specifically we want to profile all valid event types. *
*******************************************************************************

HISSERV MF=(E, serviceList), REQUEST=PROFILE, *
ACTION=START, OUTPROFILETKN=ProfToken, *
NAME=ProfName, EXITRTRN=ExitMod, SAMPLE=YES, *
EVENT=HisEvn_ValidEvnTyp, RETCODE=LRetCode, *
RSNCODE=LRsncode

* Place code to check return/reason codes here
*
* Place code to obtain storage of length HisEvnData_kLength, save address in R8, length in EvnDataAnsLen
*
EVN LOOP DS 0H

*******************************************************************************
* Query the current state of the events *
*******************************************************************************

Using HisEvnData, R8

HISSERV MF=(E, serviceList), REQUEST=QUERY, *
TYPE=EVENTDATA, PROFILETKN=ProfToken, *
CPUMASK=ALL, ANSAREA=HisEvnData, *
ANSLEN=EvnDataAnsLen, RETCODE=LRetCode, *
RSNCODE=LRsncode

ICM R5, B'11111111', LRetCode
JZ EVNDATAAREA_GOOD
LA R4, Hisserv_kRetWarn
CLR R5, R4
JE EVNDATAAREA_WARN
EVNDATAAREA_BAD DS 0H

* Place code to check bad return/reason codes here
*
EVNDATAAREA_WARN DS 0H
L R5, LRsncode
NILH R5, 0
CHI R5, Hisserv_kRsncWarn_AnsAreaSmall
JNE EVNAREA_BAD

* Place code to free storage of length EvnDataAnsLen, then obtain new storage of length HisEvnData_Length, save address in R8, length in EvnDataAnsLen
*
J EVN LOOP
EVNDATAAREA_GOOD DS 0H
```
**HISERV macro**

* Place code to process the returned events here.
*  
  L R5,EvqQueryVal
  BCT R5,EVNDONE
  ST R5,EvqQueryVal
  STIMER WAIT,BINTVL=EvnIntv
  J EVN_LOOP

EVNDONE DS 0H

* Stop profiling the system *
*  
  HISSERV MF=(E,serviceList),REQUEST=PROFILE,
  *  
  ACTION=STOP,PROFILETNK=ProfToken,
  *  
  RETCODE=LRetCode,RSNCODE=LrsnCode

* Place code to check return/reason codes here
*  
  EvnIntv DC F'6000' One minute interval between *
  event queries
  ProfName DC CL8'SAMPLE01' External name for profiler
  ExitMod DC CL8'SAMPEXRT' EXITRTN Name
  HISSYERV Return code information and *  
  ANSAREA mappings.

DynArea DSECT
LRetCode DS F
LRsnCode DS F
ProfToken DC CL16'0'
EvnQueryVal DC F'60' Query events 60 times
EvnAnsArea DS XL(HisEvn_Len+HisEvnCtr_Len)
EvnAnsLen DS AD(HisEvn_Len+HisEvnCtr_Len)
EvnDataAnsLen DS D

*  
* HISEXRTN CSECT, the EXITRTN located in LPA, LNKLIST or the *
* nucleus.
*  
* HISEXRTN CSECT
  Using HisExitParm,R1
  CLI HisExitParm_Func,HisExitParmFunc_kStat
  JNE CHECK_SMP
  PUSH USING
  USING HisStatParm,R1

* Place code to process any service actions
*
J DONE
POP USING

CHECK_SMP DS 0H
CLI HisExitParm_Func,HisExitParmFunc_kSmp
JNE DONE
PUSH USING
USING HisSmpParm,R1

* Place code to process the full SDBs
*
POP USING
DONE DS 0H
HISYEXIT
HISYMPX
HISSEerv macro
Chapter 23. HSPSERV — Read from and write to a Hiperspace

Description

HSPSERV transfers data between virtual storage areas in address spaces and hiperspaces. It reads data from a hiperspace to an address space and it writes data to a hiperspace from an address space.

A hiperspace can be either a **standard hiperspace**, of which there are two types, shared and nonshared, or an **ESO** (expanded storage only) **hiperspace**:

- The nonshared standard hiperspace and the shared standard hiperspace are backed with real storage and, if necessary, auxiliary storage. Through the buffer area in the address space, your program can view or **scroll** through the hiperspace. HSPSERV SWRITE and HSPSERV SREAD transfer data to and from a standard hiperspace. You create a standard hiperspace through the HSTYPE=SCROLL parameter on the DSPSERV macro. The description of HSPSERV macro for standard hiperspaces begins on “Read and write services for standard hiperspaces.”

- The ESO hiperspace is backed only with real storage. It is a high-speed buffer area or **cache** for data that your program needs. HSPSERV CWRITE and HSPSERV CREAD transfer data to and from an ESO hiperspace. You create an ESO hiperspace through the HSTYPE=CACHE parameter on the DSPSERV macro. The description of the HSPSERV macro for ESO hiperspaces begins on “Read and write services for ESO hiperspaces” on page 266.

The STOKEN parameter identifies the specific hiperspace to be read from or written to. The HSPALET parameter specifies an optional ALET for the hiperspace. The RANGLIST parameter identifies one or more of the storage ranges in the address space and the one or more storage ranges in the hiperspace. A storage range consists of contiguous 4K byte blocks starting on a 4K byte boundary.

HSPSERV is also described in [z/OS MVS Programming: Assembler Services Reference ABE-HSP](https://www.ibm.com/support/knowledgecenter/en/ST4KJQ_2.11.0/abe-hsp Maneuverer). With the exception of the parameters that are valid only for supervisor state or PSW key 0 through 7 programs: CREAD, CWRITE, ADDRSP, and KEEP. For more information about hiperspaces and data spaces see [z/OS MVS Programming: Extended Addressability Guide](https://www.ibm.com/support/knowledgecenter/en/ST4KJQ_2.11.0/eadg-Maneuerer).  

Read and write services for standard hiperspaces

Environment

The requirements for the caller who specifies SREAD and SWRITE are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state and any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> PASN=HASN=SASN is required for a nonshared standard hiperspace for which an ALET is not used (that is, the HSPALET parameter is omitted).</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
</tbody>
</table>
**Environmental factor**  
Locks: No locks held  
Control parameters: Must be in the caller's primary address space. If the caller's PSW key is not zero, the PSW key must match the storage key associated with the control parameters.

**Programming requirements**
- If you code the HSFALET parameter on the HSPSERV macro, you must first code the SYSSTATE macro to indicate the ASC mode of your program.
- If you code the HSFALET parameter on the HSPSERV macro, you must provide a 144-byte save area in the caller's primary address space.
- The range list must be addressable in the caller's primary address space.

**Restrictions**
If you code HSFALET, and you have an FRR recovery routine that gains control while HSPSERV is executing, your recovery routine cannot attempt retry at the time of error.

**Input register information**
Before issuing the HSPSERV macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

However, if the caller specifies the HSFALET parameter:
- General purpose register (GPR) 13 must contain the address of a 144-byte save area. The save area must be in the caller's primary address space.
- Access register (AR) 13 must contain 0, regardless of whether the caller is in primary or AR address space control (ASC) mode.

**Output register information**
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
HPSERV macro

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

The following figure describes the characteristics and restrictions for the use of standard hiperspaces, the hiperspaces that allow your program to scroll through large areas of data.
HPSERV macro

Non-shared standard hiperspace:
- For problem state and PSW key 8-F callers:
  - If an ALET is not used, the caller's TCB must own the hiperspace.
  - If an ALET is used, any TCB in the caller's home address space can own the hiperspace.
- For supervisor state or PSW key 0-7 callers, any TCB in the caller's home address space can own the hiperspace.
- If an ALET is used:
  - The ALET must be used for a hiperspace on the caller's DU-AL or PASN-AL.
  - The cross memory mode can be any.
- If an ALET is not used, the cross memory mode must be PASN=HASN.
- For PSW key 0 callers, can have any storage key and can be fetch protected.
- For PSW key 1-F callers requesting SWRITE or SREAD RELEASE=YES, must have matching storage key.
- For PSW key 1-F callers requesting SREAD RELEASE=NO, can have non-matching storage key only if hiperspace is not fetch-protected.

Shared standard hiperspace:
- Problem state and PSW key 8-F callers must use an ALET.
- Any task in the system can own the hiperspace. If the owning task is not in the caller's home or primary address space, the owner's home address space must be non-swappable.
- If an ALET is used, it must be for a hiperspace on the caller's DU-AL or PASN-AL.
- The cross memory mode can be any.
- For PSW keys 1-F callers requesting SWRITE or SREAD RELEASE=YES, must have matching storage key.
- For PSW key 1-F callers requesting SREAD RELEASE=NO, can have non-matching storage key only if hiperspace is not fetch-protected.

Syntax
The standard form of the HPSERV macro for standard hiperspaces is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
</tbody>
</table>
### HPSERV macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>One or more blanks must precede HPSERV.</td>
</tr>
<tr>
<td>HPSERV</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow HPSERV.</td>
</tr>
<tr>
<td>SREAD</td>
<td></td>
</tr>
<tr>
<td>SWRITE</td>
<td></td>
</tr>
<tr>
<td>,STOKEN=token-addr</td>
<td>token-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,HSPALET=alet-addr</td>
<td>alet-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,NUMRANGE=n</td>
<td>n: Number from 1 to 50.</td>
</tr>
<tr>
<td>,NUMRANGE=num-addr</td>
<td>num-addr: RX-type address or register (2) - (12). Default: NUMRANGE=1.</td>
</tr>
<tr>
<td>,RANGLIST=list-addr</td>
<td>list-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RELEASE=NO</td>
<td>Default: RELEASE=NO.</td>
</tr>
<tr>
<td>,RELEASE=YES</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=ret-addr</td>
<td>ret-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsn-addr</td>
<td>rsn-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=S</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

**SREAD**

Requests that the system read data from a standard hiperspace to an address space.

STOKEN and RANGLIST are required parameters on the SREAD request. HSPALET, NUMRANGE, RELEASE, RSNCODE, and RETCODE are optional parameters.

**SWRITE**

Requests that the system write data to a standard hiperspace from an address space.

**Note:**
1. When HSPSERV returns to the caller after the SWRITE operation, the contents of the address space storage range are not preserved. You can use the address space area again.

2. If the hiperspace maps a data-in-virtual object, do not issue an SWRITE request while a DIV SAVE request is current.

STOKEN and RANGLIST are required parameters on the SWRITE request. HSPALET, NUMRANGE, RETCODE, and RSNCODE are optional parameters.

\texttt{STOKEN=stoken-addr}

Specifies the address of the eight-character variable that contains the STOKEN for the standard hiperspace from which the data is to be read or into which the data is to be written. Restrictions on standard hiperspaces are described in Figure 5 on page 262.

\texttt{HSPALET=alet-addr}

Specifies either the address of a fullword or a register that contains the ALET for the hiperspace that is to be accessed. The ALET must be for a hiperspace that is on the caller's DU-AL or PASN-AL.

The HSPALET parameter is optional except for the following case: If the caller accesses a shared hiperspace, is in problem state and has PSW key 8 - F, HSPALET is required.

Use of the HSPALET parameter requires that the caller provide a 144-byte save area in the caller's primary address space. AR/GPR 13 must provide addressability to this area regardless of the caller's ASC mode. GPR 13 must contain the address of the area and AR 13 must contain 0.

If you code HSPALET, do not code RELEASE=YES.

If you code HSPALET, and you have an FRR recovery routine that gains control while HSPSERV is executing, your recovery routine cannot attempt retry at the time of error.

\texttt{NUMRANGE=n}

\texttt{NUMRANGE=num-addr}

Specifies the number of entries, from 1 to 50, or specifies a fullword that identifies the number of entries in the range list (that the RANGLIST parameter points to), or specifies a register containing the address of a fullword containing the number of entries. The default is NUMRANGE=1.

If you omit NUMRANGE, HSPSERV reads or writes one entry in the range list.

\texttt{RANGLIST=list-addr}

Specifies a fullword that contains an address of a list of ranges (up to 50) that the system is to read or write, or specifies a register that contains the address of the fullword pointer to the range list. The range list consists of a number of entries (specified by NUMRANGE) where each entry consists of three words as follows:

\textbf{First word}

The starting virtual address in the address space into which the data is to be read or from which the data is to be written.

\textbf{Second word}

The starting virtual address in the hiperspace from which the system is to read or into which the system is to write.
**HPSERV macro**

**Third word**
The number of blocks the system is to read or write. Note that the address is the block number followed by 12 binary zeros.

An example of how to code the RANGLIST parameter when NUMRANGE=3 is as follows:

```
NUMRANGE=3 ,RANGLIST=(5)
```

or

```
NUMRANGE=3, RANGLIST=RANGADDR
```

Further restrictions on the areas in the address space and the hiperspace are described in [Figure 5 on page 262](#).

On return, only if the caller issued the HPSERV macro with the HSRALET parameter, the range list values might be different from the input values if the system could not at first successfully complete the read or write operation. In that case, the system changes the range list values, but does not restore the input values when it finally returns control to the caller.

**RELEASE=NO**
**RELEASE=YES**
Specifies whether or not the system is to release the hiperspace pages after it completes the SREAD operation. RELEASE is valid only with SREAD.

RELEASE=NO specifies that the system does not release the hiperspace pages after it completes the SREAD operation. Unless a subsequent SWRITE request changes the data, the same data will be available again on the next SREAD request. RELEASE=NO is the default.

RELEASE=YES specifies that, after the SREAD request, the system is to release the storage that backed the data in the hiperspace. If you code RELEASE=YES, do not code HSRALET.

**RETCODE=ret-addr**
Specifies the location where the system is to store the return code. The return code is also in GPR 15.

**RSNCODE=rsn-addr**
Specifies the location where the system is to store the reason code. The reason code is also in GPR 0.

**MF=S**
Specifies the standard form of the macro. This form generates code to place the parameters into an inline parameter list and invoke the service.
ABEND codes

HSPSERV might abnormally terminate with abend code X‘01D’. See z/OS MVS System Codes for an explanation of abend code X‘01D’.

Return and reason codes

When control returns from HSPSERV SREAD or HSPSERV SWRITE, GPR 15 (and ret-addr, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and rsn-addr, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Note: yy is X‘09’ for SREAD and X‘0A’ for SWRITE.

Table 28. Return and Reason Codes for HSPSERV SREAD and HSPSERV SWRITE

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                      | **Meaning:** HSPSERV completed successfully.  
Action: None. |
| 08 xxxy      | xxyy05xx                | **Meaning:** System error. The system rejects the request. A hiperspace page is unavailable.  
Action: Record the return and reason code and supply it to the appropriate IBM support personnel. |
| 08 xxxy      | xxyy06xx                | **Meaning:** System error. The system rejects the request. An address space page is unavailable.  
Action: Record the return and reason code and supply it to the appropriate IBM support personnel. |
| 0C xx      | xx006xx                 | **Meaning:** System error. System failure because of environmental problems.  
Action: Record the return and reason code and supply it to the appropriate IBM support personnel. |

Read and write services for ESO hiperspaces

Environment

The requirements for the caller who requests CREAD and CWRITE are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state or PSW key 0 - 7</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller may hold locks, but is not required to hold any</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>The parameter list and range list must be in nonpageable, non-DREF storage. If the caller specifies HSPALET and is disabled, the save area must also be in nonpageable, non-DREF storage. The parameter list and save area must all be in the common area or in the private area of the caller’s primary address space.</td>
</tr>
</tbody>
</table>
**Programming requirements**

- If you code the HSPALET parameter on the HSPSERV macro, you must first code the SYSSTATE macro to indicate the ASC mode of your program.
- If you code the HSPALET parameter on the HSPSERV macro, you must provide a 144-byte save area in the caller’s primary address space.
- The range list must be addressable in the caller’s primary address space.

**Restrictions**

If you code HSPALET, and you have an FRR recovery routine that gains control while HSPSERV is executing, your recovery routine cannot attempt retry at the time of error.

**Input register information**

Before issuing the HSPSERV macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

However, if the caller specifies the HSPALET parameter:

- General purpose register (GPR) 13 must contain the address of a 144-byte save area. The save area must be in the caller’s primary address space.
- Access register (AR) 13 must contain 0, regardless of whether the caller is in primary or AR address space control (ASC) mode.

**Output register information**

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.
The following figure describes the characteristics and restrictions for the use of ESO hiperspaces, the hiperspaces that act as a high-speed buffer or cache for data.

Syntax

The standard form of the HSPSERV macro for ESO hiperspaces follows.

CAUTION:
Code the parameters on the HSPSERV CREAD and HSPSERV CWRITE macros very carefully. Read the requirements for the address space buffer and the hiperspace, as listed in Figure 6. For performance reasons, the system does not verify the location of the addresses you specify on these macros. Incorrect coding can cause damage to the system.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
</tbody>
</table>

Figure 6. Characteristics and Restrictions for ESO Hiperspaces
Syntax Description

/bslash
One or more blanks must precede HSPSERV.

HSPSERV

/bslash
One or more blanks must follow HSPSERV.

CREAD
CWRITE

,STOKEN=token-addr
stoken-addr: RX-type address or register (2) - (12).

,HSPALET=alet-addr
alet-addr: RX-type address or register (2) - (12).

,NUMRANGE=n
n: A number from 1 to 50.

,NUMRANGE=num-addr
num-addr: RX-type address or register (2) - (12). Default: NUMRANGE=1.

,RANGLIST=list-addr
list-addr: RX-type address or register (2) - (12).

,ADDRSP=HOME
Default: ADDRSP=HOME.

,ADDRSP=PRIMARY

,ADDRSP=COMMON

,KEEP=YES
Default: KEEP=YES.

,KEEP=NO

,RETCODE=ret-addr
ret-addr: RX-type address or register (2) - (12).

,RSNCODE=rsn-addr
rsn-addr: RX-type address or register (2) - (12).

,MF=S

Parameters

The parameters are explained as follows:

**CREAD**

Requests that the system read data from an ESO hiperspace.

If all blocks requested to be read are available in the hiperspace, then the system performs the read operation. However, if one or more of the blocks to be read are no longer available in the hiperspace, then the system returns a failing return code. (See return code 08.) In this case, the system does not tell you which blocks it successfully reads, if any.
HSPSERV macro

STOKEN and RANGLIST are required parameters on the CREAD request. ADDRSP, NUMRANGE, RSNCODE, and RETCODE are optional parameters.

CWRITE
Requests that the system write data to an ESO hiperspace. If the system cannot write all the requested blocks to the hiperspace, then it doesn't write any and rejects the request. (See return code 08.) In this case, the data in the specified range in the hiperspace is unpredictable. Therefore, after an unsuccessful write, do not issue another CREAD against the failing hiperspace range of virtual storage until an intervening CWRITE is successful.

STOKEN and RANGLIST are required parameters on the CWRITE request. ADDRSP, NUMRANGE, KEEP, RSNCODE, and RETCODE are optional parameters.

STOKEN=stoken-addr
Specifies the address of the 8-character variable that contains the STOKEN for the ESO hiperspace from which the data is to be read or into which the data is to be written. Restrictions on the hiperspace are described in Figure 6 on page 268.

HSPALET=alet-addr
Specifies either the address of a fullword or a register that contains the ALET for the hiperspace that is to be accessed. The ALET must be for a hiperspace that is on the caller's DU-AL or PASN-AL.

Use of the HSPALET parameter requires that the caller provide a 144-byte save area in the caller's primary address space or in the common area. If the caller is disabled, the save area must be in nonpageable storage. AR/GPR 13 must provide addressability to this area regardless of the caller's ASC mode. GPR 13 must contain the address of the area and AR 13 must contain 0.

If you code HSPALET, do not code RELEASE=YES.

If you code HSPALET and you have an FRR recovery routine that gains control while HSPSERV is executing, your recovery routine cannot attempt retry at the time of error.

NUMRANGE=n
NUMRANGE=num-addr
Specifies a fullword that identifies the number of entries in the range list (that the RANGLIST parameter points to), or specifies a register containing the address of a fullword containing the number of entries, or specifies the number of entries, from 1 to 50. The default is NUMRANGE=1.

If you omit NUMRANGE, then HSPSERV reads or writes one virtual range.

RANGLIST=list-addr
Specifies a fullword that contains the address of a parameter area in nonpageable storage that contains a list of up to 50 ranges that the system is to read or write, or specifies a register that contains the address of the fullword pointer to the range list.

The range list consists of a number of entries (specified by NUMRANGE) where each entry consists of three words as follows:

First word
The starting virtual address in the address space into which the data is to be read or from which the data is to be written.
HPSERV macro

Second word
The starting virtual address in the hiperspace from which the system is to read or into which the system is to write.

Third word
The number of blocks the system is to read or write.

An example of how to code the RANGLIST parameter when NUMRANGE=3 is as follows:

NUMRANGE=3, RANGLIST=(5)

or

NUMRANGE=3, RANGLIST=RANGADDR

The one or more address space ranges on RANGLIST must be consistent with the ADDRSP parameter. When you specify ADDRSP=COMMON, each address space range described in the range list must reside entirely within CSA and have no intersections with other common area subpools or the private area. When you specify ADDRSP=HOME or ADDRSP=PRIMARY, each address space range described in the range list must reside entirely within the private area.

Restrictions on the areas in the address space and the hiperspace are described in Figure 6 on page 268.

The range list must be in the common area or in the private area of the caller's primary address space.

,ADDRSP=HOME
,ADDRSP=PRIMARY
,ADDRSP=COMMON

Specifies the location of the virtual storage range from which the system is to read or into which the system is to write. The location can be the caller's home address space (ADDRSP=HOME), the caller's primary address space (ADDRSP=PRIMARY), or the CSA (ADDRSP=COMMON). The default is ADDRSP=HOME.

,KEEP=YES
,KEEP=NO

Specifies whether or not the system preserves the source data in the virtual storage of the address space after it completes the CWRITE request. KEEP is valid only on the CWRITE request.

If you specify KEEP=YES, the data in the specified address space is unchanged and available for reference. The default is KEEP=YES.
HPSERV macro

If you specify KEEP=NO, the system might not preserve the data in the address space. If your program will reuse the same virtual storage area after the CWRITE request completes, use KEEP=NO.

\[ \text{RETCODE}= \text{ret-addr} \]

Specifies the location where the system is to store the return code. The return code is also in GPR 15.

\[ \text{RSNCODE}= \text{rsn-addr} \]

Specifies the location where the system is to store the reason code. The reason code is also in GPR 0.

\[ \text{MF}=S \]

Specifies the standard form of the macro. This form generates code to place the parameters into an inline parameter list and invoke the macro service.

ABEND codes

HPSERV might abnormally terminate with abend code X'01D'. See [z/OS MVS System Codes](#) for an explanation of abend code X'01D'.

Return and reason codes

When control returns from HPSERV CREAD or HPSERV CWRITE, GPR 15 (and ret-addr, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and rsn-addr, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Note: yy is X'07' for CREAD and X'08' for CWRITE.

Table 29. Return and Reason Codes for HPSERV CREAD and HPSERV CWRITE

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: HPSERV completed successfully. Action: None.</td>
</tr>
<tr>
<td>08</td>
<td>xyy01xx</td>
<td>Meaning: Program error. The hiperspace data you requested is not available (CREAD request). Action: The data must be retrieved from its permanent copy.</td>
</tr>
<tr>
<td>08</td>
<td>xyy02xx</td>
<td>Meaning: Program error. The system rejects the request because an address space page is not currently backed by real storage. You can repeat the HPSERV request after you reference one or more pages, which causes the system to page the storage in CWRITE request. Action: Reference the page or pages that are not in processor storage.</td>
</tr>
<tr>
<td>08</td>
<td>xyy03xx</td>
<td>Meaning: Environmental error. The system rejects the request because the necessary real storage frames are not currently available. Action: Rerun your program one or more times during a period of lower system usage. If the problem persists, consult your system programmer, who might be able to tune the system so that more resources are available to your program.</td>
</tr>
</tbody>
</table>
### HSPSERV - List form

Use the list form of the HSPSERV macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters.

#### Syntax

The list form of the HSPSERV macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: Symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede HSPSERV.</td>
</tr>
<tr>
<td>HSPSERV</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow HSPSERV.</td>
</tr>
<tr>
<td>PLISTVER=vernum</td>
<td><em>vernum</em>: Parameter list version 0 or 1</td>
</tr>
<tr>
<td></td>
<td><strong>Default</strong>: Version that allows all specified parameters.</td>
</tr>
<tr>
<td>,MF=(L,list-addr)</td>
<td><em>list-addr</em>: Symbol.</td>
</tr>
<tr>
<td>,MF=(L,list-addr,attr)</td>
<td><em>attr</em>: 1- to 60-character input string. <strong>Default</strong>: 0D.</td>
</tr>
</tbody>
</table>
HSPSERV macro

| Parameters for the list form of HSPSERV are as follows:

```
PLISTVER=vernum
```

Specifies the macro version associated with HSPSERV. PLISTVER is an optional parameter that determines which parameter list the system generates. Specify 0 if you use parameters only from this group:

- ADDRSP
- CREAD
- CWRITE
- KEEP
- MF
- NUMRANGE
- PLISTVER
- RANGLIST
- RELEASE
- RETCODE
- RSNCODE
- SREAD
- STOKEN
- SWRITE

If you use the HSPALET parameter, specify 1.

If you do not specify PLISTVER, the default is to allow all of the parameters you specify on the invocation to be processed.

```
,MF=(L,list-addr)
,MF=(L,list-addr,attr)
```

Specifies the list form of HSPSERV.

`list-addr` is the address of the storage area for the parameter list.

`attr` is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code `attr`, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

HSPSERV - Execute form

The execute form of the HSPSERV macro changes parameters in the control parameter list that the system created through the list form of the macro and performs the specified operation.
## Syntax

The execute form of the HSPSERV macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><em>name</em>: Symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td><code>HSPSERV</code></td>
<td></td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede HSPSERV.</td>
</tr>
<tr>
<td><code>SREAD</code></td>
<td></td>
</tr>
<tr>
<td><code>SWRITE</code></td>
<td></td>
</tr>
<tr>
<td><code>CREAD</code></td>
<td></td>
</tr>
<tr>
<td><code>CWRITE</code></td>
<td></td>
</tr>
<tr>
<td><code>,STOKEN=</code></td>
<td><code>stoken-addr</code>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,HSPALET=</code></td>
<td><code>alet-addr</code>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,NUMRANGE=</code></td>
<td><code>num-addr</code>: RX-type address or register (2) - (12). Default: <code>NUMRANGE=1</code>.</td>
</tr>
<tr>
<td><code>,RANGLIST=</code></td>
<td><code>list-addr</code>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,RELEASE=</code></td>
<td><code>release-addr</code>: RX-type address or register (2) - (12). Default: <code>RELEASE=NO</code>.</td>
</tr>
<tr>
<td><code>,ADDRSP=</code></td>
<td><code>ADDRSP=HOME</code>. Default: <code>ADDRSP=HOME</code>.</td>
</tr>
<tr>
<td><code>,KEEP=</code></td>
<td><code>KEEP=YES</code>. Default: <code>KEEP=YES</code>.</td>
</tr>
<tr>
<td><code>,RETCODE=</code></td>
<td><code>ret-addr</code>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,RSNCODE=</code></td>
<td><code>rsn-addr</code>: RX-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
| `,MF=` | `list-addr`: RX-type address or register (2) - (12).
HSPSERV macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,MF=(E,list-addr,NOCHECK)</td>
<td>Default: COMPLETE.</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained under the standard form of the HSPSERV macro with the following exceptions:

,MF=(E,list-addr,COMPLETE)
,MF=(E,list-addr,NOCHECK)

Specifies the execute form of the HSPSERV macro.

list-addr specifies the area that the system uses to store the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that you did not specify.

NOCHECK specifies that the system does not check for required parameters and does not supply the optional parameters that you did not specify.

HSPSERV - Modify form

Use the modify form of the HSPSERV macro together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

Syntax

The modify form of the HSPSERV macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede HSPSERV.</td>
</tr>
<tr>
<td>HSPSERV</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow HSPSERV.</td>
</tr>
<tr>
<td>SREAD</td>
<td></td>
</tr>
<tr>
<td>SWRITE</td>
<td></td>
</tr>
<tr>
<td>CREAD</td>
<td></td>
</tr>
<tr>
<td>CWRITE</td>
<td></td>
</tr>
<tr>
<td>,STOKEN=stoken-addr</td>
<td>stoken-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,HSPALET=alet-addr</td>
<td>alet-addr: RX-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
### Parameters

Parameters for the modify form of HPSERV are described in the standard form of the macro with the following exceptions:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,NUMRANGE=1</td>
<td>Default: NUMRANGE=1.</td>
</tr>
<tr>
<td>,NUMRANGE=num-addr</td>
<td>num-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RANGLIST=list-addr</td>
<td>list-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RELEASE=NO</td>
<td>Default: RELEASE=NO.</td>
</tr>
<tr>
<td>,ADDRSP=HOME</td>
<td>Default: ADDRSP=HOME.</td>
</tr>
<tr>
<td>,ADDRSP=PRIMARY</td>
<td></td>
</tr>
<tr>
<td>,ADDRSP=COMMON</td>
<td></td>
</tr>
<tr>
<td>,KEEP=YES</td>
<td>Default: KEEP=YES.</td>
</tr>
<tr>
<td>,KEEP=NO</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=ret-addr</td>
<td>ret-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsn-addr</td>
<td>rsn-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=(E,list-addr,COMPLETE)</td>
<td>list-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=(M,list-addr,NOCHECK)</td>
<td>Default: COMPLETE.</td>
</tr>
</tbody>
</table>

### Parameters

Parameters for the modify form of HPSERV are described in the standard form of the macro with the following exceptions:

- ,NUMRANGE=1
- ,NUMRANGE=num-addr
- ,RANGLIST=list-addr
- ,RELEASE=NO
- ,ADDRSP=HOME
- ,ADDRSP=PRIMARY
- ,ADDRSP=COMMON
- ,KEEP=YES
- ,KEEP=NO
- ,RETCODE=ret-addr
- ,RSNCODE=rsn-addr
- ,MF=(E,list-addr,COMPLETE)
- ,MF=(M,list-addr,NOCHECK)

Specifies the modify form of the HPSERV macro.

- list-addr specifies the area that the system uses to store the parameters.
- COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that you did not specify.
- NOCHECK specifies that the system does not check for required parameters and does not supply the optional parameters that you did not specify.
HSPSERV macro
Chapter 24. IARCP64 — 64-bit cell pool services

Description

Use IARCP64 to request 64-bit cell pool services.

With IARCP64, you can request to:

- Build a pool (REQUEST=BUILD).
- Obtain a cell from the pool (REQUEST=GET).
- Return a cell to the pool (REQUEST=FREE).
- Delete the pool (REQUEST=DELETE).

Note: There is diagnostic support for 64 bit cell pools in IPCS via the CBFORMAT command. CBF cpid STR(IAXCPHD) formats the cell pool header, where cpid is the cell pool identifier that was returned on IARCP64 REQUEST=BUILD. If you cannot locate your cell pool identifier in the dump, simply browse storage starting at X’100000000’ and issue a FIND on CPHD. There might be multiple cell pools, so you must look at the cell contents to make sure you have the right pool. To see details about all of the cells in the pool, use the EXIT option as follows: CBF cpid STR(IAXCPHD) EXIT.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>For IARCP64 REQUEST=BUILD, use of the COMMON=YES, TYPE=DREF, TYPE=FIXED, OWNINGTASK=RCT, MEMLIMIT=NO, or MOTKN parameter or the Key00ToF0 parameter with a value other than X’90’, require any of the following:</td>
</tr>
<tr>
<td></td>
<td>• Supervisor state</td>
</tr>
<tr>
<td></td>
<td>• PSW key 0-7</td>
</tr>
<tr>
<td></td>
<td>• APF authorized</td>
</tr>
<tr>
<td></td>
<td>All other options have a minimum authorization of Problem state and PSW key 8-15. For IARCP64 REQUEST=GET, FREE or DELETE, the caller must be able to modify the storage for the cell pool. That means the caller must be in key 0 or in the same key as the cell pool or the cell pool must be in the public key (key 9). Supervisor state is required for the TRACE=YES option. APF authorization has no bearing on these services.</td>
</tr>
</tbody>
</table>

- Dispatchable unit mode: Task or SRB
- Cross memory mode: Any PASN, any HASN, any SASN
- AMODE: 64-bit
- ASC mode: Primary or access register (AR)
IARCP64 macro

Environmental factor  Requirement
Interrupt status:  For the BUILD and DELETE requests, enabled.
    For the GET and FREE requests:
        • The caller might be enabled or disabled for interrupts when requesting cells that are from pools are defined as COMMON=YES and TYPE=FIXED.
        • For all other combinations of options, the caller must be enabled for interrupts.
Locks:  For the BUILD and DELETE requests, no locks may be held.
    For the GET request, the following locks must be held by the caller or must be obtainable by IARCP64:
        • For requests with EXPAND=NO, the caller might hold locks but is not required to hold any.
        • For requests with COMMON=NO and EXPAND=YES, the caller might hold the local lock (LOCAL or CML) of the current primary address space.
        • For requests with COMMON=YES and EXPAND=YES, the locking restrictions for the caller are the same as for IARV64 REQUEST=GETCOMMON.
    For the FREE request, the caller might hold locks but is not required to hold any.
Control parameters:  Control parameters must be in the primary address space.

Programming requirements
   Specify SYSSTATE AMODE64=YES prior to invoking this macro.

Restrictions
   None.

Input register information
   Before issuing the IARCP64 macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
   When control returns to the caller, the 64-bit GPRs contain:

   For REQUEST=BUILD:

   Register  Contents
           0  Reason code in the low 32 bits if the return code is not 0. Otherwise, used as a work register by the system.
           1  Used as a work register by the system.
          2-13  Unchanged.
           14  Used as a work register by the system.
           15  Return code in the low 32 bits.

   For REQUEST=GET:
Register

Contents

0  Reason code in the low 32 bits if the return code is not 0. Otherwise, used as a work register by the system.

1  The address of the obtained cell.

2-12  Unchanged if REGS=S\-SAVE was specified, used as work registers by the system if REGS=USE was specified.

13  Unchanged.

14  Used as a work register by the system.

15  Return code in the low 32 bits.

For REQUEST=FREE:

Register

Contents

0-1  Used as a work register by the system.

2-12  Unchanged if REGS=S\-SAVE was specified, used as work registers by the system if REGS=USE was specified.

13  Unchanged.

14-15  Used as a work register by the system.

For REQUEST=DELETE:

Register

Contents

0-1  Used as a work register by the system.

2-13  Unchanged.

14-15  Used as work registers by the system.

When control returns to the caller, the ARs contain:

Register

Contents

0-1  Used as work registers by the system.

2-13  Unchanged.

14-15  Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.
### Syntax

The standard form of the IARCP64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name name:</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>△</td>
<td>One or more blanks must precede IARCP64.</td>
</tr>
<tr>
<td>IARCP64</td>
<td></td>
</tr>
<tr>
<td>△</td>
<td>One or more blanks must follow IARCP64.</td>
</tr>
<tr>
<td>REQUEST=BUILD</td>
<td></td>
</tr>
<tr>
<td>REQUEST=GET</td>
<td></td>
</tr>
<tr>
<td>REQUEST=FREE</td>
<td></td>
</tr>
<tr>
<td>REQUEST=DELETE</td>
<td></td>
</tr>
<tr>
<td>,HEADER=header</td>
<td>header: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>,CELLSIZE=cellsiz e</td>
<td>cellsiz e: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>,OUTPUT_CPID=output_cpid</td>
<td>output_cpid: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>,COMMON=NO</td>
<td></td>
</tr>
<tr>
<td>,COMMON=YES</td>
<td></td>
</tr>
<tr>
<td>,OWNINGTASK=CURRENT</td>
<td></td>
</tr>
<tr>
<td>,OWNINGTASK=MOTHER</td>
<td></td>
</tr>
<tr>
<td>,OWNINGTASK=IPT</td>
<td></td>
</tr>
<tr>
<td>,OWNINGTASK=JOBSTEP</td>
<td></td>
</tr>
<tr>
<td>,OWNINGTASK=CMRO</td>
<td></td>
</tr>
<tr>
<td>,OWNINGTASK=RCT</td>
<td></td>
</tr>
<tr>
<td>,MEMLIMIT=YES</td>
<td>Default: MEMLIMIT=YES</td>
</tr>
<tr>
<td>,MEMLIMIT=NO</td>
<td></td>
</tr>
<tr>
<td>,MOTKN=motkn</td>
<td>motkn: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>,MOTKN=NO_MOTKN</td>
<td>Default: MOTKN=NO_MOTKN</td>
</tr>
<tr>
<td>,DUMP=LIKERGN</td>
<td></td>
</tr>
<tr>
<td>,DUMP=LIKELSQA</td>
<td></td>
</tr>
</tbody>
</table>
## Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.DUMP=LIKECSA</td>
<td></td>
</tr>
<tr>
<td>.DUMP=LIKESQA</td>
<td></td>
</tr>
<tr>
<td>.DUMP=NO</td>
<td></td>
</tr>
<tr>
<td>.DUMPPRIO=dumpprio</td>
<td>dumpprio: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>.OWNER=HOME</td>
<td></td>
</tr>
<tr>
<td>.OWNER=PRIMARY</td>
<td></td>
</tr>
<tr>
<td>.OWNER=SECONDARY</td>
<td></td>
</tr>
<tr>
<td>.OWNER=SYSTEM</td>
<td></td>
</tr>
<tr>
<td>.OWNER=BYASID</td>
<td></td>
</tr>
<tr>
<td>.OWNINGASID=owningasid</td>
<td>owningasid: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>.FPROT=YES</td>
<td></td>
</tr>
<tr>
<td>.FPROT=NO</td>
<td></td>
</tr>
<tr>
<td>.TYPE=PAGEABLE</td>
<td></td>
</tr>
<tr>
<td>.TYPE=DREF</td>
<td></td>
</tr>
<tr>
<td>.TYPE=FIXED</td>
<td></td>
</tr>
<tr>
<td>.CALLERKEY=YES</td>
<td></td>
</tr>
<tr>
<td>.CALLERKEY=NO</td>
<td></td>
</tr>
<tr>
<td>.KEY00TOF0=key00tof0</td>
<td>key00tof0: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>.TRAILER=COND</td>
<td></td>
</tr>
<tr>
<td>.TRAILER=YES</td>
<td></td>
</tr>
<tr>
<td>.TRAILER=NO</td>
<td></td>
</tr>
<tr>
<td>.FAILMODE=RC</td>
<td></td>
</tr>
<tr>
<td>.FAILMODE=ABEND</td>
<td></td>
</tr>
<tr>
<td>.LOCALSYSAREA=NO</td>
<td><strong>Default:</strong> LOCALSYSAREA=NO</td>
</tr>
<tr>
<td>.LOCALSYSAREA=YES</td>
<td></td>
</tr>
<tr>
<td>.INPUT_CPID=input_cpid</td>
<td>input_cpid: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>.CELLADDR=celladdr</td>
<td>celladdr: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>.EXPAND=YES</td>
<td></td>
</tr>
</tbody>
</table>
### IARCP64 macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,EXPAND=NO</td>
<td></td>
</tr>
<tr>
<td>,TRACE=YES</td>
<td></td>
</tr>
<tr>
<td>,TRACE=NO</td>
<td></td>
</tr>
<tr>
<td>,CELLNAME=cellname</td>
<td>cellname: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>,CELLADDR=celladdr</td>
<td>celladdr: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>,REGS=SAVE</td>
<td></td>
</tr>
<tr>
<td>,REGS=USE</td>
<td></td>
</tr>
<tr>
<td>,INPUT_CPID=input_cpid</td>
<td>input_cpid: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12), (GPR15), (REG15), or (R15).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12), (GPR0), (GPR00), (REG0), (REG00), or (R0).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=0</td>
<td></td>
</tr>
<tr>
<td>,MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td>list addr: RS-type address or register (1) - (12)</td>
</tr>
<tr>
<td>,MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

- **name**
  - An optional symbol, starting in column 1, that is the name on the IARCP64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

- **REQUEST**
  - A required parameter that indicates the type of request:
    - BUILD
    - GET
    - FREE
    - DELETE
REQUEST=BUILD
Request to build the pool. The initial pool size is 1 MB. The CELLSIZE and TRAILER specifications determine how many available cells are in the pool.

REQUEST=GET
Request to obtain a cell from the pool.

REQUEST=FREE
Request to return a cell to the pool. Note that this request is unconditional and will abnormally end in the event of a problem. No return and reason codes are provided; therefore, do not specify the RETCODE and RSNCODE parameters.

REQUEST=DELETE
Request to delete the pool. Note that this request is unconditional and will abnormally end in the event of a problem. No return and reason codes are provided; therefore, do not specify the RETCODE and RSNCODE parameters.

Parameters for REQUEST=BUILD

The following parameters are valid when you specify REQUEST=BUILD:

,HEADER=header
A required input parameter that specifies information to be placed into the pool header for potential diagnostic purposes. The information helps to identify the requestor and the purpose for the pool.

To code: Specify the RS-type address, or address in register (2)-(12), of a 24-character field.

,CELLSIZE=cellsize
A required input parameter that indicates the size of a cell in the pool. The cell size can be anywhere between 1 and (1M-8192)/2 or 520,192 bytes. Cell size is rounded up to a quadword multiple for cell sizes less than a cache line. Cells larger than a cache line are rounded up to a cache line multiple. Cells larger than a page are rounded to start on a page boundary. The first cell in an extent is always located on a page boundary. Specifying a cell size that is at least 4 bytes less than the size after rounding for boundary alignment makes room for a trailer to be inserted. See TRAILER=YES below.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,OUTPUT_CPID=output_cpid
A required output parameter that is to contain the cell pool ID.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,COMMON=NO
,COMMON=YES
A required parameter that indicates if the pool is to reside in common storage.

,OWNINGTASK=CURRENT
A required parameter that indicates the task to be considered as the owner of the cell pool. When this task ends, the cell pool is automatically deleted.

 OWNINGTASK=CURRENT
 The current task is to be the owner. Do not specify this unless the program is in task mode.

 OWNINGTASK=MOTHER
 The mother task of the current task is to be the owner. If the current task is the cross-memory resource owning task, the request will fail. Do not specify this unless the program is in task mode.

 OWNINGTASK=IPT
 The initial pthread task is to be the owner. If the current task or mother task is not the IPT, then this will default to the current task as the owner. Do not specify this unless the program is in task mode.

 OWNINGTASK=JOBSTEP
 The jobstep task of the current task (the task with TCB address in field TCBJSTCB of the current task's TCB) is to be the owner. Do not specify this unless the program is in task mode.

 OWNINGTASK=CMRO
 The cross-memory resource-owning task of the current primary address space is to be the owner.

 OWNINGTASK=RCT
 The region control task (RCT) of the current primary address space is to be the owner.

 MEMLIMIT=YES
 MEMLIMIT=NO
 An optional parameter that specifies whether the 64-bit private memory objects created for this cell pool are to count towards the address space MEMLIMIT. The default is MEMLIMIT=YES.

 MEMLIMIT=YES
 The 64-bit private memory objects contribute towards the address space MEMLIMIT.

 MEMLIMIT=NO
 The 64-bit private memory objects are not counted against the address space MEMLIMIT.

 MOTKN=motkn
 MOTKN=NO_MOTKN
 An optional input parameter that identifies the memory object token to be associated with the memory object. This is expected to be a memory object token that is user-generated (as opposed to having been created by the system with the OUTMOTKN parameter of IARV64 GETSTOR). The main reason to specify your own MOTKN is to have the cell pool extents be associated with other memory objects from a dumping perspective. WARNING: If you use this MOTKN on other IARV64 REQUEST=GETSTOR calls, a call to IARCP64 REQUEST=DELETE will detach all memory objects allocated with this MOTKN. Similarly, a call to IARV64 REQUEST=DETACH with this MOTKN will result in detaching all extents of the cell pool, without deleting control
IARCP64 macro

information for the cell pool. Unpredictable behavior can result. The default is
NO_MOTKN which indicates that no memory object token is supplied to
associate this memory object with others.

To code: Specify the RS-type address, or address in register (2)-(12), of an
8-character field.

,DUMP=LIKERGN
,DUMP=LIKELSQA
,DUMP=LIKECSA
,DUMP=LIKECSA
,DUMP=NO

A required parameter that indicates how to dump this pool.

When COMMON=NO is specified:

,DUMP=LIKERGN
Dump the pool according to the rules for RGN.

,DUMP=LIKELSQA
Dump the pool according to the rules for LSQA.

,DUMP=NO
Do not dump the pool based on the RGN and LSQA SDATA options.

When COMMON=YES is specified:

,DUMP=LIKECSA
Dump the pool according to the rules for CSA.

,DUMP=LIKECSA
Dump the pool according to the rules for SQA.

,DUMP=NO
Do not dump the pool based on the CSA and SQA SDATA options.

,DUMPPRIO=dumpprio
When DUMP=LIKERGN, COMMON=NO and REQUEST=BUILD are specified,
a required input parameter that contains the dump priority to be used when
dumping the pool. The value can be in the range 1-99 with 1 being the highest
priority. See the documentation for the GETSTOR option of the IARV64 macro
for a discussion on dump priorities.

To code: Specify the RS-type address, or address in register (2)-(12), of an
one-byte field.

,OWNER=HOME
,OWNER=PRIMARY
,OWNER=SECONDARY
,OWNER=SYSTEM
,OWNER=BYASID

When COMMON=YES is specified, a required parameter that designates the
owner of the storage.

,OWNER=HOME
The home address space is to be the owner.

,OWNER=PRIMARY
The primary address space is to be the owner.

,OWNER=SECONDARY
The secondary address space is to be the owner.
IARCP64 macro

,OWNER=SYSTEM
   The system is to be the owner. Use this only when there is no specific
   address space which can be considered the owner.

,OWNER=BYASID
   The owner is the ASID specified by the OwningASID parameter.

,OWNINGASID=owningasid
   When OWNER=BYASID, COMMON=YES and REQUEST=BUILD are specified,
   a required input parameter that specifies the ASID that is to be the owner. A
   value of 0 is equivalent to having specified OWNER=SYSTEM.
   To code: Specify the RS-type address, or address in register (2)-(12), of a
   halfword field.

,FPROT=YES
,FPROT=NO
   A required parameter that indicates whether the pool storage is to be
   fetch-protected.

   ,FPROT=YES
      The pool storage is to be fetch-protected.

   ,FPROT=NO
      The pool storage is not to be fetch-protected.

,TYPE=PAGEABLE
,TYPE=DREF
,TYPE=FIXED
   A required parameter that indicates the type of storage for the pool.

   ,TYPE=PAGEABLE
      The pool storage is to be pageable.

   ,TYPE=DREF
      The pool storage is to be disabled-reference (DREF).

   ,TYPE=FIXED
      The pool storage is to be page-fixed.

,CALLERKEY=YES
,CALLERKEY=NO
   A required parameter that indicates whether the pool storage is to be in the
   key of the caller of the BUILD request.

   ,CALLERKEY=YES
      The pool storage is to be in the key of the caller.

   ,CALLERKEY=NO
      The pool storage is not to be in the key of the caller, but instead in the key
      specified by the Key00ToF0 parameter.

,KEY00TOF0=key00tof0
   When CALLERKEY=NO is specified, a required input parameter that indicates
   the key for the pool storage. The value should be in the range X'00' to X'F0'
   (i.e., the key 0-15 in the high 4 bits of the byte) for a caller that is key 0-7,
   supervisor state, or APF-authorized. The value X'90' is the only accepted key
   for a caller that is key 8-15, problem state, and not APF-authorized.
   To code: Specify the RS-type address, or address in register (2)-(12), of an
   one-byte field.

,TRAILER=COND
,TRAILER=YES
A required parameter that indicates whether the cell is to have a trailer area after the user portion of the cell which is set on GET processing and checked on FREE processing. Note that requesting a trailer can cause the cell size to be increased to provide room for the trailer. This increase in size occurs before rounding for boundary alignment. For example, requesting a cell size of 4096 and TRAILER=YES results in cells being 8192 bytes in length. If you do not need the entire 4096 bytes, specify a cell size of 4092 bytes and now the trailer fits in the same page.

TRAILER=COND
The cell storage should have trailer processing in the following cases:
- When the service-rounded cell size has room for the trailer without requiring a larger cell to be allocated.
- When system diagnostic controls requests trailers be appended to cells obtained by IARCP64. If this results in trailer processing, it will work as described for TRAILER=YES below.

Note that the system diagnostic control for trailers in IARCP64 cell pools is examined at BUILD time only.

TRAILER=YES
The cell storage is to have trailer processing. If the application writes past the end of the specified cell size, it will overrun the trailer. On a FREE request, this will be detected and cause an ABEND.

TRAILER=NO
The cell storage is not to have trailer processing, even if requested via a system diagnostic control.

FAILMODE=RC
FAILMODE=ABEND
A required parameter that indicates what to do if the request is not successful.

FAILMODE=RC
The request should return with a failure return code when there are insufficient memory resources to satisfy the request. All errors in parameter specification or parameter access will result in the request abnormally ending.

FAILMODE=ABEND
The request should abnormally end when there are insufficient memory resources to satisfy the request.

LOCALSYSAREA=NO
LOCALSYSAREA=YES
When COMMON=NO is specified, an optional parameter that specifies whether this is an explicit allocation request for 64-bit virtual storage in the local system area. This parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is LOCALSYSAREA=NO.

LOCALSYSAREA=NO
The request will not be satisfied from the local system area.

LOCALSYSAREA=YES
The request is to be satisfied from the local system area. The storage obtained with this keyword will not be copied during fork processing. The use of local system area storage does not preclude checkpoint from succeeding.
Parameters for REQUEST=GET

The following parameters are valid when you specify REQUEST=GET:

INPUT_CPID=input_cpid
A required input parameter that contains the cell pool ID returned on the successful BUILD request.
To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

CELLADDR=celladdr
An optional output parameter of the obtained cell. If CELLADDR is not specified, the cell address is left in register 1.
To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

EXPAND=YES
,EXPAND=NO
A required parameter that indicates whether to attempt expanding the pool if there is no available cell.

EXPAND=YES
Indicates that an attempt to expand the pool should be made. Each successful expansion results in a 1 MB increase in the pool size.

EXPAND=NO
Indicates that no attempt to expand the pool should be made.

TRACE=YES
,TRACE=NO
A required parameter that indicates whether the invocation is to be traced. Note that tracing is available only to supervisor state callers.

TRACE=YES
The entry is to be traced. If you are running in supervisor state, use this option, unless performance needs dictate otherwise. Note that TRACE=YES on GET also results in TRACE=YES on FREE, so if you use TRACE=YES, ensure that the FREE request is in supervisor state.

TRACE=NO
The entry is not to be traced. You must use this option if running in problem state.

FAILMODE=RC
,FAILMODE=ABEND
A required parameter that indicates what to do if the request is not successful.

FAILMODE=RC
The request should return with a failure return code when there are insufficient memory resources to satisfy the request. All errors in parameter specification or parameter access will result in the request abnormally ending.

FAILMODE=ABEND
The request should abnormally end when there are insufficient memory resources to satisfy the request.

REGS=SAVE
,REGS=USE
A required parameter that indicates how to deal with the registers.
The request should save and preserve the contents of 64-bit GPRs 2 - 12, starting at offset 40 in a 144-byte area pointed to by register 13.

The request may use registers 2 - 12.

### Parameters for REQUEST=FREE

The following parameters are valid when you specify REQUEST=FREE:

- **CELLNAME=cellname**
  - A required input parameter that identifies the cell to free.
  - **To code:** Specify the RS-type address, or address in register (2)-(12), of a character field.

- **CELLADDR=celladdr**
  - The address of the cell to free.
  - **To code:** Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

- **REGS=SAVE**
- **REGS=USE**
  - A required parameter that indicates how to deal with the registers.

- **REGS=SAVE**
- **REGS=USE**
  - The request should save and preserve the contents of 64-bit GPRs 2 - 12, starting at offset 40 in a 144-byte area pointed to by register 13.

- **REGS=USE**
  - The request may use registers 2 - 12.

### Parameters for REQUEST=DELETE

The following parameters are valid when you specify REQUEST=DELETE:

- **INPUT_CPID=input_cpid**
  - A required input parameter that contains the cell pool ID returned on the BUILD request.
  - **To code:** Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

### Optional parameters

The following parameters are optional:

- **RETCODE=retcode**
  - An optional output parameter into which the return code is to be copied from GPR 15. If you specify (GPR15), (REG15), or (R15), the value will be left in GPR 15.
  - **To code:** Specify the RS-type address of a fullword field, or register (2)-(12), (GPR15), (REG15), or (R15).
IARCP64 macro

\[RSNCODE=rsncode\]
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify (GPR0), (GPR00), (REG0), (REG00), or (R0), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12), (GPR0), (GPR00), (REG0), (REG00), or (R0).

\[PLISTVER=IMPLIED\_VERSION\]
\[PLISTVER=MAX\]
\[PLISTVER=0\]
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- IMPLIED\_VERSION, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED\_VERSION is the default.
- MAX, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
  
  If you can tolerate the size change, specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- 0, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED\_VERSION
- MAX
- A decimal value of 0

\[MF=S\]
\[MF=(L,\ \text{list addr})\]
\[MF=(L,\ \text{list addr,attr})\]
\[MF=(L,\ \text{list addr,0D})\]
\[MF=(E,\ \text{list addr})\]
\[MF=(E,\ \text{list addr,COMPLETE})\]
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF= S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.
IARCP64 macro

\texttt{list \ addr}

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

\texttt{attr}

An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code \texttt{attr}, the system provides a value of 0D.

\texttt{COMPLETE}

This parameter specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

The IARCP64 caller might receive abend code X’DC4’. For detailed abend code information, see \textit{z/OS MVS System Codes}.

Return and reason codes

When the IARCP64 macro returns control to your program:

- GPR 15 (and \texttt{retcode}, when you code \texttt{RETCODE}) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and \texttt{rsncode}, when you code \texttt{RSNCODE}) contains a reason code.

Macro IAXSERVC provides equated symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equated symbol associated with each reason code.

\textit{Table 30. Return and Reason Codes for the IARCP64 Macro}

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None</td>
<td>\texttt{IARCP64Rc_OK} Meaning: IARCP64 request successful. Action: None required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{BUILD} Meaning: Cell pool built Action: None required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{DELETE} Meaning: Cell pool deleted and storage freed Action: None required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{GET} Meaning: Cell returned to the pool. Action: None required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\texttt{FREE} Meaning: Cell returned to the pool. Action: None required.</td>
</tr>
<tr>
<td>04</td>
<td>None</td>
<td>\texttt{IARCP64Rc_Warn} Meaning: Warning Action: Refer to the action provided with the specific reason code.</td>
</tr>
</tbody>
</table>
### Table 30. Return and Reason Codes for the IARCP64 Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>xx0400xx</td>
<td>Equate Symbol: IARCP64RsnGetOutOfCells</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The request to the IARCP64 GET service specified EXPAND=NO and the current extent is out of cells.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Either change the request to specify EXPAND=YES or write logic to deal with no cell being available.</td>
</tr>
<tr>
<td>08</td>
<td>None</td>
<td>Equate Symbol: IARCP64Rc_Fail</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> Service failed due to running out of resources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Refer to the action provided with the specific reason code.</td>
</tr>
<tr>
<td>08</td>
<td>xx0401xx</td>
<td>Equate Symbol: IARCP64RsnMemlimitExhausted</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The request to either the IARCP64 BUILD, IARCP64 GET when the pool is being expanded or the IARST64 GET when a new extent is required was not able to obtain private storage due to the address space MEMLIMIT.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Either raise the MEMLIMIT of the address space or determine if private storage is being consumed excessively somewhere.</td>
</tr>
<tr>
<td>08</td>
<td>xx0402xx</td>
<td>Equate Symbol: IARCP64Rsn64BitCommonExhausted</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The request to either the IARCP64 BUILD, IARCP64 GET when the pool is being expanded or the IARST64 GET when a new extent is required was not able to obtain common storage due to there being insufficient 64 bit common storage to satisfy the request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> For common storage, either raise the system limit on common (HVCOMMON) or determine if common storage is being consumed excessively somewhere.</td>
</tr>
</tbody>
</table>

### Examples

1. Build a pool according to the following specifications:
   - Cells 32-bytes long
   - In private storage
   - With an owning task of the current task
   - Dumped similar to "RGN" processing
   - Not fetch-protected
   - Pageable storage
   - In key 3
   - Provide a diagnostic trailer. Note that requesting a diagnostic trailer causes the cell size to internally be rounded up from 32 bytes to 48 bytes
   - Provide return code if the request is not successful

The coding sample follows:

```assembly
IARCP64 REQUEST=BUILD,HEADER=theHeader,
            CELLSIZE=theCellsSize,OUTPUT_CPID=theCPID,
            COMMON=NO,OWNINGTASK=CURRENT,DUMP=LIKE RGN,
            FPROT=NO,TYPE=PAGEABLE,
            CALLERKEY=NO,KEY00TOF0=theKEY,
```
2. Obtain a cell from the pool.
   • Do not expand the pool if no cell is available
   • Provide Return Code if the request is not successful
   • Save and restore registers
   The coding sample follows:
   
   IARCP64 REQUEST=GET, INPUT_CPID=theCPID, 
   CELLADDR=theCellAddr, EXPAND=NO, FAILMODE=RC, REGS=SAVE, 
   RETCODE=LRETCODE, RSNCODE=LRSNCODE, 

   (Place code to check return/reason codes here.)

   IAXSERVC Return/Reason code information
   DYNAREA DSECT
   LRETCODE DS F
   LRSNCODE DS F
   theCPID DS D
   IARCP64 MF=(L,IARCP64L)

3. Free a cell.
   • Save and restore registers
   The coding sample follows:
   
   IARCP64 REQUEST=FREE, 
   CELLADDR=theCellAddr, REGS=SAVE

   IAXSERVC Return/Reason code information
   DYNAREA DSECT
   theCPID DS D
   theCellAddr DS D

4. Delete the pool.
   The coding sample follows:
   
   IARCP64 REQUEST=DELETE, INPUT_CPID=theCPID, 
   MF=(L,IARCP64L)

   IAXSERVC Return/Reason code information
   DYNAREA DSECT
   theCPID DS D
   IARCP64 MF=(L,IARCP64L)
IARCP64 macro
Chapter 25. IARR2V — Convert a central storage address to a virtual storage address

Description

Use the IARR2V macro to convert a central storage address to a virtual storage address. This conversion can be useful when you have the central storage address from handling I/O or doing diagnostic support and need to know the corresponding virtual address.

When the input storage address is a central storage address that backs a single page, the system returns the ASID that indicates the address space that owns the central storage, and the STOKEN that indicates the address space or data space that uses the central storage. When a central storage address does not back any page, or backs a read-only nucleus page, the system returns a non-zero return code and reason code.

For more information on the use of the IARR2V macro, see z/OS MVS Programming: Authorized Assembler Services Guide. IARR2V is also described in the z/OS MVS Programming: Assembler Services Guide with the exception of the LINKAGE parameter.

Environment

The requirements for the caller are:

- Environmental factor: Problem state with any PSW key. For the LINKAGE parameter, supervisor state with any PSW key.
- Dispatchable unit mode: Task or SRB
- Cross memory mode: Any PASN, any HASN, any SASN
- AMODE: 24-, 31- or 64-bit.
- ASC mode: Primary or access register (AR)
- Interrupt status: Enabled or disabled for I/O and external interrupts
- Locks: The caller may hold the local or CPU lock, but is not required to hold any locks.
- Control parameters: None.

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the IARR2V macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:
IARR2V macro

Register

Contents

0  ASID if return code is 0 or 4; otherwise, reason code. The ASID value is X'FFFF' if the returned virtual address represents common storage.

1  Virtual storage address if return code is 0 or 4; otherwise, used as a work register by the system.

2-13  Unchanged.

14  Used as a work register by the system.

15  Return code.

When control returns to the caller, the ARs contain:

Register

Contents

0  First four bytes of STOKEN if return code is 0 or 4; otherwise, used as a work register by the system.

1  Last four bytes of STOKEN if return code is 0 or 4; otherwise, used as a work register by the system.

2-13  Unchanged.

14  Total shared view count if return code is 0 or 4; otherwise, used as a work register by the system.

15  Valid shared view count if return code is 0 or 4; otherwise, used as a work register by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IARR2V macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARR2V.</td>
</tr>
<tr>
<td>IARR2V</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARR2V.</td>
</tr>
<tr>
<td>RSA=rsa_addr</td>
<td>rsa_addr: RS-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>RSA64=rsa_addr64</td>
<td>rsa_addr64: RS-type address, or register (2) - (12).</td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

\textbf{RSA=rsa\_addr}

Specifies the name (RS-type) or address (in register 2-12) of an input fullword that contains the central storage address to be converted to a virtual storage address. This keyword is used to provide a 31-bit real address. RSA and RSA64 are mutually exclusive keywords. You must specify one or the other.

\textbf{RSA64=rsa\_addr64}

Specifies the name (RS-type) or address (in register 2-12) of an input double-word that contains the central storage address to be converted to a virtual storage address. This keyword is used to provide a 64-bit real address. RSA and RSA64 are mutually exclusive keywords. You must specify one or the other. To use this keyword, the SYSTATE macro must be invoked specifying ARCHLVL greater than 1.

\textbf{VSA=vsa\_addr}

Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword that the system uses to return the virtual storage address that corresponds to the input central storage address.

\textbf{VSA64=vsa\_addr64}

Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword that the system uses to return the 64-bit virtual storage address that
IARR2V macro

corresponds to the input central storage address. VSA and VSA64 are mutually exclusive keywords. To use this keyword, the SYSTATE macro must be invoked specifying ARCHLVL greater than 1.

,ASID=asid_addr
Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword that the system uses to return the ASID of the address space associated with the output virtual storage address. The system returns the ASID in bits 16-31 of the fullword, and clears bits 1-15 to 0. If the input central storage address backs a page that is shared through the use of the IARVSE RV macro, the system sets bit 0 to 1; otherwise, bit 0 contains 0.

,STOKEN=stoken_addr
Specifies the name (RS-type) or address (in register 2-12) of an optional 8-character output field that the system uses to return the STOKEN for the address space or data space associated with the output virtual storage address.

,WORKREG=work_reg
,WORKREG=NONE
Specifies whether the system is to return a page sharing view count. If you want the system to return a page sharing view count, specify work-reg as a digit from 2 through 12 that identifies a GPR/AR pair that the system can use as work registers. WORKREG=work_reg is required if you code NUMVIEW or NUMVALID.

WORKREG=NONE is the default and specifies that the system is not to return the sharing count.

,NUMVIEW=view_addr
Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword that the system uses to return the number of page sharing views associated with the input central storage address. This number is non-zero only if the system sets bit 0 of the ASID. NUMVIEW=view_addr is required with the WORKREG=work_reg parameter.

,NUMVALID=val_addr
Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword that the system uses to return the number of valid page sharing views associated with the input central storage address. A valid page must be currently defined in central storage. This number is non-zero only if the system sets bit 0 of the asid_addr. NUMVALID=val_addr is required with the WORKREG=work_reg parameter.

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
Specifies whether the system is to use a program call (LINKAGE=SYSTEM) or branch entry (LINKAGE=BRANCH). LINKAGE=SYSTEM is the default.

,RETCODE=retcode
Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword into which the system copies the return code from GPR 15.

,RSNCODE=rsncode
Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword into which the system copies the a reason code from GPR 0.

ABEND codes
None.
Return and reason codes

When the IARR2V macro returns control to your program, GPR 15 (and retcode if you coded RETCODE) contains the return code. If the return code is not 0 or 4, GPR 0 (and rsncode if you coded RSNCODE) contains the reason code.

Table 31. Return and Reason Codes for the IARR2V Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | None                    | **Meaning:** The IARR2V request completed successfully. The address returned in the VSA parameter represents an address space page.  
**Action:** None required. |
| 04                      | None                    | **Meaning:** The IARR2V request completed successfully. The address returned in the VSA parameter represents a data space page.  
**Action:** None required. |
| 08                      | xx001xx                 | **Meaning:** Program error. The IARR2V request was unsuccessful because the input central storage address was not within the bounds of central storage.  
**Action:** Check your input central storage address and rerun the program. |
| 08                      | xx002xx                 | **Meaning:** Program error. The IARR2V request was unsuccessful because the frame corresponding to the input central storage address was not assigned to a page.  
**Action:** Check your input central storage address and rerun the program. |
| 08                      | xx003xx                 | **Meaning:** Program error. The IARR2V request was unsuccessful because the frame corresponding to the input central storage address contains shared data, but no virtual address for any accessible address space (either home, primary, or secondary) corresponds to the frame.  
**Action:** Check your input central storage address and rerun the program. |
| 08                      | xx004xx                 | **Meaning:** System error. The IARR2V request was recursively invoked.  
**Action:** Record the return code and reason code and supply them to the appropriate IBM support personnel. |
| 08                      | xx005xx                 | **Meaning:** Program error. The IARR2V request was unsuccessful because the frame corresponding to the input central storage address was assigned, but the data space STOKEN could not be found.  
**Action:** Check your input central storage address and rerun the program. |
| 08                      | xx006xx                 | **Meaning:** Program error. The IARR2V request was unsuccessful because the virtual address is above 2G and the caller did not specify VSA64.  
**Action:** Specify VSA64 on the IARR2V invocation. |

Example 1

Convert the central storage address in variable VSA and place the result in variable VSAOUT.
IARR2V macro

LRA 1,VSA
LR 5,1
INVOKE1 IARR2V RSA=(5),VSA=VSAOUT
  
VSA    DS  F
VSAOUT DS  F

Example 2
Same as Example 1, but return ASID in variable ASIDO.
INVOKE2 IARR2V RSA=(5),ASID=ASIDO
  
ASIDO  DS  F

Example 3
Same as Example 1, but return STOKEN in variable STOKO.
INVOKE3 IARR2V RSA=(5),STOKEN=STOKO
  
STOKO  DS  F

Example 4
Obtain the total and valid number of page sharing views associated with the input address. WORKREG is required.
INVOKE4 IARR2V RSA=(5),WORKREG=(6),NUMVIEW=VIEWS,NUMVALID=VALS
  
VIEWS  DS  F
VALS   DS  F
Chapter 26. IARST64 — 64-bit storage services

Description

Use IARST64 to request 64-bit Storage Services.

With IARST64, you can request services to:

- Obtain storage (REQUEST=GET)
- Return storage (REQUEST=FREE)

Note: There is diagnostic support for 64 bit cell pools, created by IARST64, in IPCS via the CBFORMAT command. In order to locate the cell pool of interest you need to follow the pointers from HP1, to HP2, to the CPHD. For common storage, the HP1 is located in the ECVT. CBF ECVT formats the ECVT, then does a FIND on HP1. Extract the address of the HP1 from the ECVT and CBF addrhp1 STR(HP1) formats the HP1. Each entry in the HP1 represents an attribute set (storage key, storage type(pageable, DREF, FIXED), and Fetch-Protection (ON or OFF)). The output from this command contains CBF commands for any connected HP2s. Select the CBF command of interest and run it to format the HP2. The HP2 consists of pointers to cell pool headers for different sizes. Choose the size of interest and select the command that looks like this to format the cell pool header:

```
CBF addrchphd STR(IAXCPHD)
```

To see details about all of the cells in the pool, use the EXIT option as follows:

```
CBF addrchphd STR(IAXCPHD) EXIT
```

For private storage, the HP1 is anchored in the STCB. The quickest way to locate the HP1 is to run the SUMMARY FORMAT command for the address space of interest. Locate the TCB that owns the storage of interest and then scroll down to the formatted STCB. The HP1 field contains the address of the HP1. From here, the processing is the same as described for common storage above.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Use of the COMMON=YES, TYPE=DREF, TYPE=FIXED, OWNINGTASK=RCT, or the Key00ToF0 parameter with a value other than 9 requires the caller to be running in key 0-7. Use of MEMLIMIT=NO requires key 0-7 or supervisor state. All other options have a minimum authorization of problem state and PSW key 8-15.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
</tbody>
</table>
| Interrupt status       | • The caller may be enabled or disabled for interrupts when requesting storage that is defined as COMMON=YES and TYPE=DREF or TYPE=FIXED.  
                         | • For all other parameter combinations, the caller must be enabled for interrupts. |
Environmental factor
Locks: For the GET request, the following locks may be held by the caller or must be obtainable by IARST64:
  • For requests with COMMON=NO, the locking restrictions are the same as for IARV64 REQUEST=GETSTOR.
  
For the FREE request, the caller might hold locks but is not required to hold any.

Control parameters: Control parameters must be in the primary address space.

Programming requirements
None.

Restrictions
None.

Input register information
When REGS=SAVE is not used, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the IARST64 macro with REGS=SAVE, the caller must ensure that the following GPR contains the specified information:

Register   Contents
13          Address of a 144-byte area within which the 88 bytes beginning at offset 40 may be modified.

Before issuing the IARST64 macro, the caller does not have to place any information into any access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the 64-bit GPRs contain:

For REQUEST=GET

Register   Contents
0           Reason code in the low 32 bits if the return code is not 0. Otherwise, used as a work register by the system.
1           The address of the obtained storage.
2-12        Unchanged if REGS=SAVE was specified, used as work registers by the system if REGS=USE was specified.
13          Unchanged.
14          Used as a work register by the system.
15          Return code in the low 32 bits.

For REQUEST=FREE
Register

Contents

0-1  Used as a work register by the system.

2-12
  • Unchanged, if REGS=SAVE was specified.
  • Used as work registers by the system, if REGS=USE was specified.

13  Unchanged.

14-15  Used as a work register by the system.

When control returns to the caller, the ARs contain:

Register

Contents

0-1  Used as work registers by the system.

2-13  Unchanged.

14-15  Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IARST64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARST64.</td>
</tr>
<tr>
<td>IARST64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARST64.</td>
</tr>
<tr>
<td>REQUEST=GET</td>
<td></td>
</tr>
<tr>
<td>REQUEST=FREE</td>
<td></td>
</tr>
<tr>
<td>_SIZE=size</td>
<td>size: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>_AREAADDR=areaaddr</td>
<td>areaaddr: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>_COMMON=NO</td>
<td></td>
</tr>
</tbody>
</table>
### IARST64 macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,COMMON=YES</td>
<td></td>
</tr>
<tr>
<td>,OWNINGTASK=CURRENT</td>
<td></td>
</tr>
<tr>
<td>,OWNINGTASK=MOTHER</td>
<td></td>
</tr>
<tr>
<td>,OWNINGTASK=IPT</td>
<td></td>
</tr>
<tr>
<td>,OWNINGTASK=JOBSTEP</td>
<td></td>
</tr>
<tr>
<td>,OWNINGTASK=CMRO</td>
<td></td>
</tr>
<tr>
<td>,OWNINGTASK=RCT</td>
<td></td>
</tr>
<tr>
<td>,MEMLIMIT=YES</td>
<td>Default: MEMLIMIT=YES</td>
</tr>
<tr>
<td>,MEMLIMIT=NO</td>
<td></td>
</tr>
<tr>
<td>,LOCALSYSAREA=NO</td>
<td>Default: LOCALSYSAREA=NO</td>
</tr>
<tr>
<td>,LOCALSYSAREA=YES</td>
<td></td>
</tr>
<tr>
<td>,OWNER=HOME</td>
<td></td>
</tr>
<tr>
<td>,OWNER=PRIMARY</td>
<td></td>
</tr>
<tr>
<td>,OWNER=SECONDARY</td>
<td></td>
</tr>
<tr>
<td>,OWNER=SYSTEM</td>
<td></td>
</tr>
<tr>
<td>,OWNER=BYASID</td>
<td></td>
</tr>
<tr>
<td>,OWNINGASID=owningasid</td>
<td>owningasid: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>,FPROT=YES</td>
<td></td>
</tr>
<tr>
<td>,FPROT=NO</td>
<td></td>
</tr>
<tr>
<td>,TYPE=PAGEABLE</td>
<td></td>
</tr>
<tr>
<td>,TYPE=DREF</td>
<td></td>
</tr>
<tr>
<td>,TYPE=FIXED</td>
<td></td>
</tr>
<tr>
<td>,CALLERKEY=YES</td>
<td></td>
</tr>
<tr>
<td>,CALLERKEY=NO</td>
<td></td>
</tr>
<tr>
<td>,KEY00TOF0=key00tof0</td>
<td>key00tof0: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>,FAILMODE=RC</td>
<td></td>
</tr>
<tr>
<td>,FAILMODE=ABEND</td>
<td></td>
</tr>
<tr>
<td>,REGS=SAVE</td>
<td></td>
</tr>
<tr>
<td>,REGS=USE</td>
<td></td>
</tr>
</tbody>
</table>
Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,AREANAME=areaname</td>
<td>areaname: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>,AREAADDR=areaaddr</td>
<td>areaaddr: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>,REGS=SAVE</td>
<td></td>
</tr>
<tr>
<td>,REGS=USE</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12), (GPR15), (REG15), or (R15).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12), (GPR0), (GPR00), (REG0), (REG00), or (R0).</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

name
- An optional symbol, starting in column 1, that is the name on the IARST64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=GET
REQUEST=FREE
- A required parameter that indicates the type of request.

REQUEST=GET
- This parameter gets storage.

REQUEST=FREE
- This parameter returns storage.

Note:
- This request is unconditional, and will abnormally end if there is a problem. No return and reason codes are provided, so do not specify the RETCODE and RSNCODE parameters.

,SIZE=size
- When REQUEST=GET is specified, a required input parameter that indicates the size of the storage to be obtained. The size can be anywhere between 1 and 128K bytes. The size is rounded up to a power of 2. So cell sizes are 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16,384, 32,768, 65,536 and 131,072 bytes. The smallest cell size that contains the request is used. If the requested size is at least 4 bytes less than the rounded up cell size, a trailer will be added to check for storage overruns. For storage that is larger than what IARCP64 supports, consider using IARCP64 or IARV64 GETSTOR or GETCOMMON. Do not specify a value exceeding 128K or incorrect results may ensue.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,AREAADDR=areaaddr
- When REQUEST=GET is specified, an optional output parameter, of the obtained storage. If AREAADDR is not specified, the cell address is left in register 1.
IARST64 macro

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,COMMON=NO
,COMMON=YES
When REQUEST=GET is specified, a required parameter that indicates if the pool is to reside in common storage.

,COMMON=NO
This parameter indicates that the pool is not to reside in common storage.

,COMMON=YES
This parameter indicates that the pool is to reside in common storage.

,OWNINGTASK=CURRENT
,OWNINGTASK=MOTHER
,OWNINGTASK=IPT
,OWNINGTASK=JOBSTEP
,OWNINGTASK=CMRO
,OWNINGTASK=RCT
When COMMON=NO and REQUEST=GET are specified, a required parameter that indicates the task that is to be considered the owner.

,OWNINGTASK=CURRENT
This parameter indicates that the current task is to be the owner. Do not specify this unless the program is in task mode.

,OWNINGTASK=MOTHER
This parameter indicates that the mother task of the current task is to be the owner. If the current task is the cross-memory resource owning task, the request will fail. Do not specify this unless the program is in task mode.

,OWNINGTASK=IPT
This parameter indicates that the initial pthread task (subtask running under Unix System Services) is to be the owner. If the current task or mother task is not the IPT, then this will default to the current task as the owner. Do not specify this unless the program is in task mode.

,OWNINGTASK=JOBSTEP
This parameter indicates that the jobstep task of the current task (the task with TCB address in field TCBJSTCB of the current task’s TCB) is to be the owner. Do not specify this unless the program is in task mode.

,OWNINGTASK=CMRO
This parameter indicates that the cross-memory resource-owning task is to be the owner.

,OWNINGTASK=RCT
This parameter indicates that the region control task (RCT) is to be the owner.

,MEMLIMIT=YES
,MEMLIMIT=NO
When COMMON=NO and REQUEST=GET are specified, an optional parameter that indicates whether MEMLIMIT applies if an additional 1M segment is obtained to satisfy the request. The default is MEMLIMIT=YES.

,MEMLIMIT=YES
This parameter indicates that MEMLIMIT applies.
IARST64 macro

MEMLIMIT=NO
This parameter indicates that MEMLIMIT does not apply.

LOCALSYSAREA=NO
LOCALSYSAREA=YES
When Common=NO and request=GET are specified, an optional parameter that specifies whether this is an explicit allocation request for 64-bit virtual storage in the local system area. The LOCALSYSAREA parameter can be used only by callers running in supervisor state or with a PSW key 0-7. THE DEFAULT IS LOCALSYSAREA=NO.

LOCALSYSAREA=NO
The request will not be satisfied from the local system area.

LOCALSYSAREA=YES
The request is to be satisfied from the local system area. The storage obtained with this keyword will not be copied during Fork processing. The use of local system area storage does not preclude checkpoint or restart from succeeding.

OWNER=HOME
OWNER=PRIMARY
OWNER=SECONDARY
OWNER=SYSTEM
OWNER=BYASID
When COMMON=YES and REQUEST=GET are specified, a required parameter that designates the owner of the storage.

OWNER=HOME
This parameter indicates that the home address space is to be the owner.

OWNER=PRIMARY
This parameter indicates that the primary address space is to be the owner.

OWNER=SECONDARY
This parameter indicates that the secondary address space is to be the owner.

OWNER=SYSTEM
This parameter indicates that the system is to be the owner. Use this only when there is no specific address space which can be considered the owner.

OWNER=BYASID
This parameter indicates that the owner is the ASID specified by the OwningASID parameter.

OWNINGASID=owningasid
When OWNER=BYASID, COMMON=YES and REQUEST=GET are specified, a required input parameter that specifies the ASID that is to be the owner. A value of 0 is equivalent to having specified OWNER=SYSTEM. Do not specify a value exceeding 32767 or incorrect results may ensue.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

FPROT=YES
FPROT=NO
When REQUEST=GET is specified, a required parameter that indicates if the pool storage is to be fetch-protected.
IARST64 macro

,FPRT=YES
   This parameter indicates that the pool storage is to be fetch-protected.

,FPRT=NO
   This parameter indicates that the pool storage is not to be fetch-protected.

,TYPE=PAGEABLE
,TYPE=DREF
,TYPE=FIXED
   When REQUEST=GET is specified, a required parameter that indicates the type of storage for the pool.

,TYPE=PAGEABLE
   This parameter indicates that the pool storage is to be pageable.

,TYPE=DREF
   This parameter indicates that the pool storage is to be disabled-reference (DREF).

,TYPE=FIXED
   This parameter indicates that the pool storage is to be page-fixed.

,CALLERKEY=YES
,CALLERKEY=NO
   When REQUEST=GET is specified, a required parameter that indicates if the pool storage is to be in the key of the caller of the GET request.

,CALLERKEY=YES
   This parameter indicates that the pool storage is to be in the key of the caller.

,CALLERKEY=NO
   This parameter indicates that the pool storage is not to be in the key of the caller, but instead in the key specified by the Key00ToF0 parameter.

,KEY00TOF0=key00tof0
   When CALLERKEY=NO and REQUEST=GET are specified, a required input parameter that indicates the key for the pool storage. The value should be in the range x'00' to x'F0' (i.e., the key 0-15 in the high 4 bits of the byte) for a caller that is key 0. For caller's in key 1-7, you can only request storage that is the same as the CALLERKEY, so there is no reason to use this parameter unless you request key x'90'. The value x'90' is the only accepted key for a caller that is key 8-15. Be sure that the value is a multiple of 16 within the required range or incorrect results may ensue.

   To code: Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

,FAILMODE=RC
,FAILMODE=ABEND
   When REQUEST=GET is specified, a required parameter that indicates what to do if the GET request is not successful due to out of memory in the requested area conditions.

,FAILMODE=RC
   This parameter returns with a failure return code.

   Note: There will be cases for which an ABEND occurs regardless of the specification of FAILMODE=RC.

,FAILMODE=ABEND
   This parameter abnormally ends.
When REQUEST=GET is specified, a required parameter that indicates how to deal with the registers.

**REGS=SAVE**
This parameter saves and preserves the contents of 64-bit GPRs 2 - 12 starting at offset 40 in a 144 byte area pointed to by register 13.

**REGS=USE**
This parameter indicates that you may use registers 2 - 12.

**AREANAME=areaname**
**AREAADDR=areaaddr**
When REQUEST=FREE is specified, a required input parameter.

**AREANAME=areaname**
A parameter that is the area to free.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a character field.

**AREAADDR=areaaddr**
A parameter that contains the address of the area to free.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

When REQUEST=FREE is specified, a required parameter that indicates how to deal with the registers.

**REGS=SAVE**
This parameter saves and preserves the contents of 64-bit GPRs 2 - 12 starting at offset 40 in a 144 byte area pointed to by register 13.

**REGS=USE**
This parameter indicates that you may use registers 2 - 12.

**RETCODE=retcode**
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, or R15 (within or without parentheses), the value will be left in GPR15.

**To code:** Specify the RS-type address of a fullword field, or register (2)-(12), (GPR15), (REG15), or (R15).

**RSNCODE=rsncode**
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR0.

**To code:** Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (0), (GPR0), (GPR00), (REG0), (REG00), or (R0).

### ABEND codes

The IARST64 caller might receive abend code X'DC4'. For detailed abend code information, see z/OS MVS System Codes.
In the following IARST64 abend reason codes, the bytes designated "xx" are for diagnostic purposes and have no significance to the external interface. Equate IARST64AbendRsnCodeMask has been provided to let you build a mask to ignore those bytes.

<table>
<thead>
<tr>
<th>Hexadecimal Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx0410xx</td>
<td>Equate Symbol: IARST64AbendRsnCellAddrLow</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning:</strong> The storage address passed to the IARST64 FREE service is within a megabyte used for storage pools, but the address is less than the address of the 1st usable storage address.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Correct the address passed to IARST64 FREE, making sure it is the same address that was returned from IARST64 GET.</td>
</tr>
<tr>
<td>xx0413xx</td>
<td>Equate Symbol: IARST64AbendRsnCellNotInExtent</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning:</strong> The request was to the IARCP64 or IARST64 FREE service and the address of the storage passed in, is not within the bounds of a cell pool.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> The address passed to IARST64 REQUEST=FREE must be the same as the address obtained from IARST64 REQUEST=GET.</td>
</tr>
<tr>
<td>xx0419xx</td>
<td>Equate Symbol: IARST64AbendRsnCellOverRun</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning:</strong> The request was to the IARCP64 or IARST64 FREE service and the trailer data at the end of the cell was detected as being overrun. If the overrun is sufficiently large, it will cause damage to the following cell. The caller is abnormally ended so they can fix the code to not use more storage than requested.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Determine whether the storage has been overrun or whether the trailer data was overlaid by some other code. Fix the code so it only uses the amount of storage requested.</td>
</tr>
<tr>
<td>xx041Axx</td>
<td>Equate Symbol: IARST64AbendRsnCellNotInUse</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning:</strong> The request was to the IARCP64 or IARST64 FREE service and the address of the storage passed in, is already in the freed state. This will happen when an application frees the storage twice.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> Determine whether the current application is freeing the storage twice or whether it is using a cell that some other storage is freeing twice.</td>
</tr>
<tr>
<td>xx041Bxx</td>
<td>Equate Symbol: IARST64AbendRsnNotOnCellBoundary</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning:</strong> The request was to the IARCP64 or IARST64 FREE service and the address of the storage passed in, is not on a cell boundary in the cell pool from which the GET request was satisfied.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> When freeing storage with IARST64 REQUEST=FREE, make sure to specify the address that was returned by IARST64 REQUEST=GET.</td>
</tr>
</tbody>
</table>
**Hexadecimal Reason Code | Equate Symbol Meaning and Action**

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx041Cxx</td>
<td>Equate Symbol: IARST64AbendRsnIARV64Error</td>
<td>During processing of IARST64 GET, a call to the IARV64 service for GETSTOR, GETCOMMON, PAGEFIX or PROTECT failed. The failing return code from IARV64 was placed in register 2 prior to the abend. The failing reason code from IARV64 was placed in register 3 prior to the abend.</td>
<td>Examine the return and reason code as documented under IARV64 to determine if the problem is one that you can resolve.</td>
</tr>
<tr>
<td>xx0420xx</td>
<td>Equate Symbol: IARST64AbendRsnCpheaNotQueue</td>
<td>The cell pool header authorized area was not queued to the owning task as expected. This could happen due to storage overlays or the caller bypassing the IARST64 macro and PCing directly to the service with incorrect input parameters.</td>
<td>Make sure the application is using the IARST64 macro to request storage. If the problem persists, collect a dump and contact IBM Service.</td>
</tr>
<tr>
<td>xx0425xx</td>
<td>Equate Symbol: IARST64AbendRsnPoolNotInCallerKey</td>
<td>The request to IARST64 GET was against a storage pool that was not in the key of the caller. Normally this will abend with an 0C4, but if the pool is out of cells and is in storage that is not fetch-protected, the pool expand routine verifies that the caller may modify this storage pool.</td>
<td>You must be in a key that has the ability to modify the pool storage for the request to be processed.</td>
</tr>
<tr>
<td>xx0426xx</td>
<td>Equate Symbol: IARST64AbendRsnPrimaryExtentOverlaid</td>
<td>The request to IARST64 or IARCP64 GET was against a storage pool where the primary extent control information has been overlaid.</td>
<td>Collect a dump and report the problem to IBM.</td>
</tr>
<tr>
<td>xx0427xx</td>
<td>Equate Symbol: IARST64AbendRsnSecondaryExtentOverlaid</td>
<td>The request to IARST64 or IARCP64 GET was against a storage pool where the secondary extent control information has been overlaid.</td>
<td>Collect a dump and report the problem to IBM.</td>
</tr>
<tr>
<td>xx0428xx</td>
<td>Equate Symbol: IARST64AbendRsnUnexpectedError</td>
<td>During processing of IARST64 GET an unexpected abend occurred. An SDUMP should have been generated.</td>
<td>Collect the dump and report the problem to IBM.</td>
</tr>
</tbody>
</table>
## Hexadecimal Reason Code Equate Symbol Meaning and Action

<table>
<thead>
<tr>
<th>Hexadecimal Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx0511xx</td>
<td>IARST64AbendRsnKeyGT7Common</td>
<td>The request to IARST64 GET was for common storage, but the requested or caller was greater than key 7. You cannot allocate common storage in key 8 or above.</td>
<td>Correct the key passed to IARST64 GET or change your request to get private storage.</td>
</tr>
<tr>
<td>xx0512xx</td>
<td>IARST64AbendRsnGetMotherFromCmro</td>
<td>The request was to the IARST64 GET service and specified OWNINGTASK(MOTHER), but the caller is running on the CMRO task. You can't request the mother task be the storage owner from the CMRO task.</td>
<td>Either specify CMRO as the owner or specify RCT if you want the storage to persist across termination of the CMRO.</td>
</tr>
<tr>
<td>xx0514xx</td>
<td>IARST64AbendRsnGetNotRctOrCmro</td>
<td>The request was to the IARST64 GET service for private storage and the caller was running in cross memory mode or SRB mode. In these environments the OWNINGTASK parameter must be set to RCT or CMRO. Neither of these was specified, so the request is failed.</td>
<td>Specify the OWNINGTASK parameter as RCT or CMRO.</td>
</tr>
<tr>
<td>xx0515xx</td>
<td>IARST64AbendRsnGetCellSizeZero</td>
<td>The request was to the IARST64 GET service and specified a length of zero.</td>
<td>Specify a length between 1 and 128K.</td>
</tr>
<tr>
<td>xx0516xx</td>
<td>IARST64AbendRsnGetNotAuth</td>
<td>The request was to the IARST64 GET service and specified a parameter that requires the caller to be running in key 0-7. The caller is not authorized to use authorized options of COMMON, DREF, FIXED, OWNINGTASK(RCT), CALLERKEY(NO) and Key00ToF0 set to a system key.</td>
<td>Either run the code in key 0-7 or do not use authorized options.</td>
</tr>
<tr>
<td>xx0517xx</td>
<td>IARST64AbendRsnGetCellSizeTooBig</td>
<td>The request was to the IARST64 GET service and specified a length greater than 128K.</td>
<td>Specify a size between 1 and 128K. If larger storage is needed, consider using IARCP64 or IARV64 GETSTOR or GETCOMMON.</td>
</tr>
</tbody>
</table>
### Return and reason codes

When the IARST64 macro returns control to your program:

- GPR 15 (and retcode, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNCODE) contains a reason code.

<table>
<thead>
<tr>
<th>Hexadecimal Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
</table>
| xx0518xx                | Equate Symbol: IARST64AbendRsnGetKeyNot9  
  **Meaning:** The request was to the IARST64 GET service and specified a CALLERKEY(NO) and a value for Key00ToF0 that was not key 9 and the caller is not authorized.  
  **Action:** The only key that an unauthorized user can specify is key 9. Either request key 9 or change the specification to CALLERKEY(YES). |
| xx0529xx                | Equate Symbol: IARST64AbendRsnGetSizeTooBig  
  **Meaning:** The call to the IARST64 GET service specified a cell size larger than the maximum size supported.  
  **Action:** Specify a size between 1 and 128K. If a larger storage area is needed, consider using IARCP64 or IARV64 REQUEST=GETSTOR or GETCOMMON. |
| xx052Axx                | Equate Symbol: IARST64AbendRsnValidationError  
  **Meaning:** The call to the IARST64 GET service detected a validation error when locating the storage pool to be used. Possible cause is storage overlay of the storage pool control block in the caller’s key.  
  **Action:** Collect a dump and report the problem to IBM. |
| xx052Bxx                | Equate Symbol: IARST64AbendRsnMemLimitNoUnauth  
  **Meaning:** The call to the IARST64 GET service requested MEMLIMIT=NO, but is running unauthorized (key 8-15 and problem program state).  
  **Action:** Either specify MEMLIMIT=YES or call from an authorized environment. |
| xx052Cxx                | Equate Symbol: IARST64AbendRsnCellLT4Gig  
  **Meaning:** The call to the IARCP64 or IARST64 FREE service was passed a cell address less than 4 Gig, so it can’t possibly be a valid cell address in a 64 bit cell pool.  
  **Action:** Only pass a storage address that was obtained with IARCP64 or IARST64 GET. |
| xx052Dxx                | Equate Symbol: IARST64AbendRsnLocalSysAreaYesUnauth  
  **Meaning:** The call to the IARST64 GET service requested LOCALSYSAREA=YES, but is running unauthorized (key 8-15 and problem program state).  
  **Action:** Either specify LOCALSYSAREA=NO or CALL from an authorized environment. |
IARST64 macro

Macro IAXSERVC provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code.

**Table 32. Return and Reason Codes for the IARST64 Macro**

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None</td>
<td></td>
<td>IARST64Rc_OK</td>
<td>None required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IARST64 request successful.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>None</td>
<td>IARST64Rc_Fail</td>
<td>Service failed due to running out of resources.</td>
<td>Refer to the action provided with the specific reason code.</td>
</tr>
<tr>
<td>08 xx0401xx</td>
<td></td>
<td>IARST64RsnMemlimitExhausted</td>
<td>The request to the IARST64 GET service was not able to obtain storage due to address space limits.</td>
<td>Either raise the MEMLIMIT of the address space or determine if private storage is being consumed excessively somewhere.</td>
</tr>
<tr>
<td>08 xx0402xx</td>
<td></td>
<td>IARST64Rsn64BitCommonExhausted</td>
<td>The request to the IARST64 GET service was not able to obtain storage due to system limits.</td>
<td>For common storage, either raise the system limit on common (HVCOMMON) or determine if common storage is being consumed excessively somewhere.</td>
</tr>
<tr>
<td>08 xx0403xx</td>
<td></td>
<td>IARST64RsnMemlimitZero</td>
<td>The request to IARST64 GET was not able to obtain private storage due to the address space MEMLIMIT being set to zero.</td>
<td>Either set the MEMLIMIT of the address space to a non-zero value or if authorized, specify MEMLIMIT=NO on the IARST64 GET call to tell the service to bypass the address space MEMLIMIT.</td>
</tr>
</tbody>
</table>

**Examples**

**Example 1:** Obtain storage.

Operations:
- 32-byte area
- In private storage
- With an owning task of the current task
- Dumped similar to "LSQA" processing (triggered by DREF or FIXED)
- Fetch-protected
- DREF storage
IARST64 macro

- In Key 7
- Provide Return Code if the request is not successful
- Save and restore registers

The code is as follows:

```asm
IARST64 REQUEST=GET,
    AREAADDR=theAreaAddr,
    SIZE=theAreaSize,
    COMMON=NO,OWNINGTASK=CURRENT,
    DUMP=LIKELSQA,FPROT=YES,TYE=DREF,
    CALLERKEY=NO,KEY00TOF0=theKEY,
    FAILMODE=RC,
    REGS=SAVE,
    RETCODE=LRETCODE,RSNCODE=LRSNCODE,

(Place code to check return code or reason codes here.)
    theAreaSize  DC F'32'
    theKey       DC X'70'
IAXSERVC
    DYNAREA      DSECT
    LRETCODE     DS F
    LRSNCODE     DS F
    theAreaAddr  DS D
```

**Example 2:**  Free the storage.

Operation: Save and restore registers.

The code is as follows:

```asm
IARST64 REQUEST=FREE,
    AREAADDR=theAreaAddr,
    REGS=SAVE,

(There is no return code or reason code from
IARST64 REQUEST=FREE.)

IAXSERVC
    DYNAREA      DSECT
    LRETCODE     DS F
    LRSNCODE     DS F
    theAreaAddr  DS D
```
IARST64 macro
Chapter 27. IARSUBSP — Create and delete a subspace

Description

Use the IARSUBSP macro to create and delete subspaces. A subspace is a section of address space private area storage that you have set up to contain and protect a program and its data. Subspaces provide isolation between multiple programs running in a single address space by allowing a program that runs in the subspace to reference only certain storage in the address space private area. For more information about subspaces and how to use them, see z/OS MVS Programming: Extended Addressability Guide.

Use the IARSUBSP macro to:
- Identify storage to be assigned to a subspace (IDENTIFY parameter)
- Create a subspace (CREATE parameter)
- Assign the identified storage to the created subspace (ASSIGN parameter)
- Disassociate the identified storage from the created subspace (UNASSIGN parameter)
- Delete a subspace (DELETE parameter)
- Make the storage ineligible to be assigned to a subspace (UNIDENTIFY parameter).

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>For the ASSIGN and UNASSIGN parameters, problem state with any PSW key. For IDENTIFY, CREATE, DELETE, and UNIDENTIFY, supervisor state or PSW key 0 - 7.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>For IDENTIFY, CREATE, DELETE, and UNIDENTIFY, task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>For ASSIGN and UNASSIGN, task or SRB. For IDENTIFY, CREATE, DELETE, and UNIDENTIFY, PASN=HASN=SASN. For ASSIGN and UNASSIGN, PASN=HASN, any SASN.</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller cannot hold locks.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

Before issuing IARSUBSP, the caller must obtain storage for the subspace by using the STORAGE or GETMAIN macro. See the RANGLIST parameter description for the required attributes of this storage. The caller must not release this storage until after issuing IARSUBSP UNIDENTIFY.
IARSUBSP macro

Restrictions
None.

Input register information
Before issuing the IARSUBSP macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A reason code, if GPR 15 contains a non-zero return code; otherwise, used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>2 - 13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>A return code.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 1</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>2 - 13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14 - 15</td>
<td>Used as a work register by the system.</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
The standard form of the IARSUBSP macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARSUBSP.</td>
</tr>
<tr>
<td>IARSUBSP</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARSUBSP.</td>
</tr>
</tbody>
</table>
### Syntax Description

Valid parameters (required parameters are underlined):

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDENTIFY</td>
<td>RANGLIST,NUMRANGE</td>
</tr>
<tr>
<td>CREATE</td>
<td>NAME,STOKEN,GENNAME,OUTNAME</td>
</tr>
<tr>
<td>ASSIGN</td>
<td>RANGLIST,STOKEN,NUMRANGE</td>
</tr>
<tr>
<td>UNASSIGN</td>
<td>RANGLIST,STOKEN,NUMRANGE</td>
</tr>
<tr>
<td>DELETE</td>
<td>STOKEN</td>
</tr>
<tr>
<td>UNIDENTIFY</td>
<td>RANGLIST,NUMRANGE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,RANGLIST=ranglist_addr</td>
<td>RS-type address, or address in register (2) - (12).</td>
</tr>
<tr>
<td>,NUMRANGE=numrange_addr</td>
<td>RS-type address, or address in register (2) - (12).</td>
</tr>
<tr>
<td>,NAME=name_addr</td>
<td>RS-type address, or address in register (2) - (12).</td>
</tr>
<tr>
<td>,GENNAME=NO</td>
<td>Default: GENNAME=NO</td>
</tr>
<tr>
<td>,GENNAME=COND</td>
<td></td>
</tr>
<tr>
<td>,GENNAME=YES</td>
<td></td>
</tr>
<tr>
<td>,OUTNAME=outname_addr</td>
<td>RS-type address, or address in register (2) - (12).</td>
</tr>
<tr>
<td>,STOKEN=stoken_addr</td>
<td>RS-type address, or address in register (2) - (12).</td>
</tr>
</tbody>
</table>

### Parameters

The IDENTIFY, CREATE, ASSIGN, UNASSIGN, DELETE, and UNIDENTIFY parameters designate the services of the IARSUBSP macro, and are mutually exclusive.

The parameters are explained as follows:

**IDENTIFY**

Identifies the ranges of storage specified on the RANGLIST parameter as eligible to be assigned to a subspace. When the IDENTIFY function successfully completes, the storage specified on the RANGLIST parameter cannot be referenced by a program running in a subspace until that storage is assigned to that subspace.

When you issue the IARSUBSP macro with IDENTIFY, you must specify the RANGLIST parameter. The NUMRANGE parameter is optional.

**CREATE**

Requests that the system create a subspace, and return an STOKEN by which a program can identify the subspace.
IARSUBSP macro

When you issue the IARSUBSP macro with CREATE, the NAME and STOKEN parameters are required. The GENNAME and OUTNAME parameters are optional.

**ASSIGN**
Requests that the system associate the range of storage specified on the RANGLIST parameter with the subspace indicated by the STOKEN parameter. When the range of storage has been assigned to the subspace, a program can reference the storage by issuing the BSG instruction.

When you issue the IARSUBSP macro with ASSIGN, you must specify the STOKEN and RANGLIST parameters. The NUMRANGE parameter is optional.

**UNASSIGN**
Requests that the system disassociate the storage identified by the RANGLIST parameter from the subspace identified by the STOKEN parameter. When the request is complete, the range of storage cannot be referenced by a program running in a subspace.

When you issue the IARSUBSP macro with UNASSIGN, you must specify the STOKEN and RANGLIST parameters. The NUMRANGE parameter is optional.

**DELETE**
Requests that the system delete the subspace indicated by the STOKEN parameter. The subspace can be deleted only by the task that created it.

When you issue the IARSUBSP macro with DELETE, you must specify the STOKEN parameter. Do not code any other parameters.

**UNIDENTIFY**
Identifies the ranges of storage specified on the RANGLIST parameter as ineligible to be assigned to a subspace.

If a range of storage specified on the RANGLIST parameter is still assigned to a subspace, the system will perform the UNASSIGN function before performing the UNIDENTIFY function.

When you issue the IARSUBSP macro with the UNIDENTIFY parameter, you must specify the RANGLIST parameter. The NUMRANGE parameter is optional.

\[\text{RANGLIST} = \text{ranglist_addr}\]
Specifies the address of a fullword input variable containing the address of the range list. The range list is a list of 8-byte entries in contiguous storage that indicate the ranges of storage to be:

- Made eligible or ineligible to be assigned to a subspace, when specified with the IDENTIFY or UNIDENTIFY functions
- Associated with or disassociated from a subspace, when specified with the ASSIGN or UNASSIGN functions

Each entry in the range list is 2 fullwords long. The first fullword contains the address of the beginning of the range of storage. The second fullword contains the number of 4-kilobyte (4096 bytes) pages that comprise the range of storage.

When RANGLIST is specified with the IDENTIFY or UNIDENTIFY parameter, the address in the first fullword must begin on a segment boundary. A segment is 1 megabyte (1,048,576 bytes) long. The value of the second fullword must be a multiple of 256.
IARSUBSP macro

When RANGLIST is specified with the ASSIGN or UNASSIGN parameters and the storage specified is above 16 megabytes, the requirements for the range list entries are the same as when RANGLIST is specified with IDENTIFY or UNIDENTIFY.

When RANGLIST is specified with the ASSIGN or UNASSIGN parameters and the storage specified is below 16 megabytes, the address in the first fullword must begin on a page boundary. A page is 4096 bytes. The value of the second fullword indicates the number of pages below 16 megabytes that are to be assigned to a subspace.

Each storage range must reside in a single subpool.

Obtain your subspace storage by selecting a storage subpool with the storage attributes that subspaces require. The chapter on virtual storage in z/OS MVS Programming: Authorized Assembler Services Guide contains a table listing all subpools and the storage attributes associated with them. The following are the required and optional storage attributes for subspaces.

Table 33. Storage Attributes Required for Subspaces

<table>
<thead>
<tr>
<th>Storage Attribute</th>
<th>Requirement</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Private</td>
<td>Subspace storage must be in high private or low private storage.</td>
</tr>
<tr>
<td>Fetch Protection</td>
<td>None</td>
<td>Subspace storage can be fetch-protected, but fetch-protection is not required.</td>
</tr>
<tr>
<td>Type</td>
<td>Pageable</td>
<td>Subspace storage must be pageable.</td>
</tr>
<tr>
<td>Owner</td>
<td>Task or job step</td>
<td>Subspace storage must be owned by the task creating the subspace, or a task higher in the task hierarchy.</td>
</tr>
<tr>
<td>Storage key</td>
<td>None</td>
<td>Subspace storage has no storage key requirements.</td>
</tr>
</tbody>
</table>

RANGLIST is a required parameter when you specify the IDENTIFY, UNIDENTIFY, ASSIGN, and UNASSIGN parameters.

\texttt{\textbackslash{\textasciitilde}}\textit{NUMRANGE}=numrange\_addr

Specifies the address of an optional fullword input variable that indicates the number of ranges in the range list. The number of ranges must be a least 1 and no more than 16. If you do not code NUMRANGE, the default number of ranges is 1, and the range list is limited to one entry.

NUMRANGE is an optional parameter when you specify the IDENTIFY, UNIDENTIFY, ASSIGN, or UNASSIGN parameters.

\texttt{\textbackslash{\textasciitilde}}\textit{NAME}=name\_addr

Specifies the address of the 8-byte variable or constant that contains the name of the subspace.

Subspace names are from 1 to 8 bytes long. They can contain letters, numbers, and @, #, and $, but they cannot contain embedded blanks. Names that are fewer than 8 bytes must be left-justified and padded on the right with blanks. Unless you specify GENNAME=YES, the subspace name must begin with a letter or an @, #, or $ character. When you do not code GENNAME, or you specify either GENNAME=NO or GENNAME=COND, the name cannot begin with a number or be blank.

Subspace names must be unique within the home address space of the owning task. No two subspaces can have the same name. To ensure that the names for
IARSUBSP macro

your subspaces are unique, code the GENNAME parameter to have the system
generate a unique name. If you choose to let the system generate the subspace
names for you, you must still supply three characters for the system to use.

NAME is a required parameter when you specify the CREATE parameter.

,GENNAME=NO
,GENNAME=COND
,GENNAME=YES

Specifies whether you want the system to generate a name for the subspace to
ensure that all names are unique within the address space. The system
generates a name by adding a 5-character prefix to the first three characters of
the name you supply on the NAME parameter. For example, if you supply
‘XYZDATA’ on the NAME parameter, the name becomes ‘ccccXYZ’. “cccc” is
the 5-character string generated by the system, and XYZ comes from the name
you supplied on NAME.

The keywords that are valid for GENNAME and their meanings follow:

GENNAME=NO
    The system does not generate a name. You must supply a name unique
    within the address space. GENNAME=NO is the default.

GENNAME=COND
    The system generates a unique name only if you supply a name that is
    already being used. Otherwise, the system uses the name you supply.

GENNAME=YES
    The system takes the name you supply on the NAME parameter and
    makes it unique. When you specify GENNAME=YES, the name you
    supply in the name parameter can begin with a numeric.

If you want the system to return the unique name it generates, use the
OUTNAME parameter.

GENNAME is an optional parameter when you specify the CREATE
parameter.

,OUTNAME=outname_addr

Specifies the address of the 8-byte variable into which the system returns the
subspace name it generated, if you specify GENNAME=YES or
GENNAME=COND. The OUTNAME parameter is optional when you specify
the CREATE parameter.

,STOKEN=stoken_addr

Specifies the address of the 8-byte STOKEN for the subspace. The system
returns an STOKEN value as output for a CREATE request. For other requests,
you supply this returned value as input. STOKEN is a required parameter for
the CREATE, ASSIGN, UNASSIGN, and DELETE requests.

ABEND codes

IARSUBSP might abnormally end with abend code X'3C6'. See z/OS MVS System
Codes for an explanation and programmer response.

Return and reason codes

When the IARSUBSP macro returns control to your program, GPR 15 contains one
of the following hexadecimal return codes. GPR 0 contains one of the following
hexadecimal reason codes.
### Table 34. Return and Reason Codes for the IARSUBSP Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | None                    | **Meaning:** The IARSUBSP request completed successfully.  
                          |                          | **Action:** None required. |
| 04                      | xx0115xx                | **Meaning:** IARSUBSP IDENTIFY completed successfully, but some ranges of storage had already been identified.  
                          |                          | **Action:** None required. However, you might want to take some action based on your application. |
| 04                      | xx0315xx                | **Meaning:** IARSUBSP ASSIGN completed successfully, but some of the storage specified on the RANGLIST parameter already had been assigned to the subspace indicated by the STOKEN parameter.  
                          |                          | **Action:** None required. However, you might want to take some action based on your application. |
| 04                      | xx0415xx                | **Meaning:** IARSUBSP UNASSIGN completed successfully, but one of the following conditions is true for some of the storage specified on the RANGLIST parameter:  
                          |                          | - Some storage already had been disassociated from the subspace by a previous UNASSIGN request  
                          |                          | - Some storage never had been assigned to a subspace.  
                          |                          | **Action:** None required. However, you might want to take some action based on your application. |
| 04                      | xx0615xx                | **Meaning:** IARSUBSP UNIDENTIFY completed successfully, but one of the following conditions is true for some of the storage specified on the RANGLIST parameter:  
                          |                          | - Some storage already had been made ineligible to be assigned to the subspace by a previous UNIDENTIFY request  
                          |                          | - Some storage never had been made eligible to be assigned to a subspace.  
                          |                          | **Action:** None required. However, you might want to take some action based on your application. |
| 08                      | xx0212xx                | **Meaning:** Environmental error. IARSUBSP CREATE failed. The system’s set of generated names for subspaces has been temporarily exhausted.  
                          |                          | **Action:** Reissue IARSUBSP CREATE, specifying a unique name on the NAME parameter and GENNAME=NO. Or, issue IARSUBSP UNASSIGN and IARSUBSP DELETE for any subspaces that are no longer required, and reissue the CREATE request. |
| 08                      | xx0213xx                | **Meaning:** Program error. IARSUBSP CREATE failed. The name specified on the NAME parameter is not unique within the address space.  
                          |                          | **Action:** Change the name specified on the NAME parameter to a unique name, or specify GENNAME=COND or GENNAME=YES, and reissue the request. |
| 08                      | xxFF00xx                | **Meaning:** Environmental error. IARSUBSP failed. The system does not support subspaces.  
                          |                          | **Action:** Contact your system programmer to determine if the subspace group facility can be made available. |
### IARSUBSP macro

#### Table 34. Return and Reason Codes for the IARSUBSP Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C</td>
<td>xx0114xx</td>
<td><strong>Meaning:</strong> Environmental error. IARSUBSP IDENTIFY failed. The system cannot perform any subspace services because of a shortage of resources. <strong>Action:</strong> Reissue the request. If the problem persists, contact your system programmer.</td>
</tr>
<tr>
<td>0C</td>
<td>xx0214xx</td>
<td><strong>Meaning:</strong> Environmental error. IARSUBSP CREATE failed. The system cannot perform any subspace services because of a shortage of resources. <strong>Action:</strong> Reissue the request. If the problem persists, contact your system programmer.</td>
</tr>
<tr>
<td>0C</td>
<td>xx0314xx</td>
<td><strong>Meaning:</strong> Environmental error. IARSUBSP ASSIGN failed. The system cannot perform any subspace services because of a shortage of resources. <strong>Action:</strong> Reissue the request. If the problem persists, contact your system programmer.</td>
</tr>
<tr>
<td>0C</td>
<td>xx0411xx</td>
<td><strong>Meaning:</strong> System error. IARSUBSP UNASSIGN failed. One or more pages of storage were not processed. <strong>Action:</strong> Reissue the request. If the problem persists, record the return and reason codes and supply them to the appropriate IBM support personnel.</td>
</tr>
</tbody>
</table>

#### Example

For a complete example of creating, managing, and deleting subspaces, see the chapter on subspaces in [z/OS MVS Programming: Extended Addressability Guide](https://www.ibm.com).  

### IARSUBSP - List form

Use the list form of the IARSUBSP macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

The list form of the IARSUBSP macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARSUBSP.</td>
</tr>
<tr>
<td>IARSUBSP</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARSUBSP.</td>
</tr>
<tr>
<td>MF=(L,list addr)</td>
<td><em>list addr</em>: symbol.</td>
</tr>
<tr>
<td>MF=(L,list addr,attr)</td>
<td><em>attr</em>: 1- to 60-character input string.</td>
</tr>
<tr>
<td>MF=(L,list addr,0D)</td>
<td>Default: 0D</td>
</tr>
</tbody>
</table>
Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>

Parameters

\[
\begin{align*}
\text{MF} &= (L, \text{list addr}) \\
\text{MF} &= (L, \text{list addr}, \text{attr}) \\
\text{MF} &= (L, \text{list addr}, 0 D)
\end{align*}
\]

Specifies the list form of the IARSUBSP macro.

- \text{list addr} is the name of a storage area to contain the parameters.
- \text{attr} is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code \text{attr}, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

IARSUBSP - Execute form

Use the execute form of the IARSUBSP macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The execute form of the IARSUBSP macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{name}</td>
<td>\text{name}: symbol. Begin \text{name} in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARSUBSP.</td>
</tr>
<tr>
<td>IARSUBSP</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARSUBSP.</td>
</tr>
</tbody>
</table>

Valid parameters (required parameters are underlined):

- IDENTIFY, RANGLIST, NUMRANGE
- CREATE, NAME, STOKEN, GENNAME, OUTNAME
- ASSIGN, RANGLIST, STOKEN, NUMRANGE
- UNASSIGN, RANGLIST, STOKEN, NUMRANGE
- DELETE, STOKEN
- UNIDENTIFY, RANGLIST, NUMRANGE
- \text{,RANGLIST} = \text{ranglist_addr} RS-type address, or address in register (2) - (12).
- \text{,NUMRANGE} = \text{numrange_addr} RS-type address, or address in register (2) - (12).

Default: 1 range
### IARSUBSP macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,NAME=name_addr</td>
<td>RS-type address, or address in register (2) - (12).</td>
</tr>
<tr>
<td>,GENNAME=NO</td>
<td><strong>Default:</strong> GENNAME=NO</td>
</tr>
<tr>
<td>,GENNAME=COND</td>
<td></td>
</tr>
<tr>
<td>,GENNAME=YES</td>
<td></td>
</tr>
<tr>
<td>,OUTNAME=outname_addr</td>
<td>RS-type address, or address in register (2) - (12).</td>
</tr>
<tr>
<td>,STOKEN=stoken_addr</td>
<td>RS-type address, or address in register (2) - (12).</td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td><em>list addr:</em> RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td><strong>Default:</strong> COMPLETE</td>
</tr>
<tr>
<td>,MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>

The parameters are explained under the standard form of the IARSUBSP macro with the following exception:

- ,MF=(E,list addr)
- ,MF=(E,list addr,COMPLETE)
- ,MF=(E,list addr,NOCHECK)

Specifies the execute form of the IARSUBSP macro.

- list addr specifies the area that the system uses to contain the parameters.
- COMPLETE, which is the default, specifies that the macro is to check for required parameters and supply defaults for omitted optional parameters.
- NOCHECK specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.
Chapter 28. IARVSERV — Request to share virtual storage

Description

Use the IARVSERV macro to define virtual storage areas to be shared by programs. This sharing can reduce the amount of processor storage required and the I/O necessary to support many applications that process large amounts of data. It also provides a way for programs executing in 24 bit addressing mode to access data residing above 16 megabytes.

Using IARVSERV allows programs to share data in virtual storage without the central storage constraints and processor overhead of other methods of sharing data. The type of storage access is controlled so that you can choose to allow read only or writing to the shared data with several variations. The type of storage access is called a view. Data to be shared is called the source. The source is the original data or the virtual storage that contains the data to be shared. This data is made accessible through an obtained storage area called the target. The source and target form a sharing group.

Through the IARVSERV macro, you can:

- Request that a virtual storage area (source) be eligible to be shared through a target view (SHARE parameter).
- Request that the source and targets no longer be shared (UNSHARE parameter).
- Request that the type of storage access to the data be changed.

See z/OS MVS Programming: Authorized Assembler Services Guide for more information about sharing data through the use of the IARVSERV macro. IARVSERV is also described in z/OS MVS Programming: Assembler Services Reference ABE-HSP.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state with PSW key that allows access to the source, target, or both, depending on the value specified through the TARGET_VIEW parameter. If the value specified on the NUMRANGE parameter is greater than 16, supervisor state or PSW key 0-7 is required. See z/OS MVS Programming: Authorized Assembler Services Guide for additional information.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN.</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR).</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller may hold the local lock, but is not required to hold any locks.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>
IARVSERV macro

Programming requirements

- You must specify a range list that is mapped by the IARVRL macro. This is done using the RANGLIST parameter.
- If you specify more than 16 ranges, you must put the range list in fixed storage.
- The address space owing the source or targets must be swapped in when IARVSERV is issued if either the source or target area is:
  - in an address space other than the home address space of the caller, or
  - in a data space owned by an address space other than the home address of the caller.

The address space must remain swapped in until the IARVSERV macro has completed.
- Before your program issues the IARVSERV macro, it must use the GETMAIN, STORAGE, or DSPSERV macro to obtain storage for the source, target, or both.
- Attributes for storage depend on the subpool specified on the GETMAIN, STORAGE, or DSPSERV macros. See [z/OS MVS Programming: Authorized Assembly Services Guide](#) for information on virtual storage management and subpool attributes. The following table shows the permitted combinations of storage attributes supported for the source and target areas (with the exceptions as noted in "Restrictions").

<table>
<thead>
<tr>
<th>Source Area</th>
<th>Target Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pageable</td>
<td>Pageable</td>
</tr>
<tr>
<td>Fixed in non-swappable storage with central</td>
<td>Any kind of storage</td>
</tr>
<tr>
<td>storage below 16 megabytes</td>
<td></td>
</tr>
<tr>
<td>Fixed in non-swappable storage with central</td>
<td>Any storage that does not require the backing</td>
</tr>
<tr>
<td>storage above 16 megabytes</td>
<td>of central storage below 16 megabytes (if fixed)</td>
</tr>
<tr>
<td>Fixed in swappable storage</td>
<td>Any kind of storage, provided that TARGET_VIEW=UNIQUEWRITE parameter is specified</td>
</tr>
</tbody>
</table>

Restrictions

The following restrictions apply:
- For the SHARE parameter, the source area must not contain pages in the nucleus (read-only, extended read-only, read-write and extended read-write areas).
- For the SHARE parameter, the target area must not contain page-protected or page-fixed pages.
- For the UNSHARE parameter, the sharing group must not contain page-protected pages unless the RETAIN=YES parameter is specified. The sharing group must also not contain any page-fixed pages.
- The TPROT instruction cannot be used to determine whether the invoker has write access to views in a share group with unique-write views or to a target-write view. The TPROT instruction will indicate that these views are protected when the invoker has write access.

Input register information

Before issuing the IARVSERV macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.
Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 contains a non-zero return code; otherwise, used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Return code.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

Take care when using the RETAIN=YES parameter value. With RETAIN=YES, storage is not returned to the system which reduces the amount available to the system and other programs, thus potentially affecting system performance.

In order to expedite the return of all internal control blocks for the shared storage back to the system, IBM recommends issuing IARVSERV UNSHARE against all views for both source and target that are originally shared. For an example of how to code the UNSHARE parameter, see Z/OS MVS Programming: Assembler Services Reference IAR-XCT.

Syntax

The standard form of the IARVSERV macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARVSERV.</td>
</tr>
<tr>
<td>IARVSERV</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARVSERV.</td>
</tr>
</tbody>
</table>
### IARVSERV macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHARE</td>
<td>Requests that the source be made shareable through the target to create a sharing group. When you issue the IARVSERV macro with SHARE, you must specify the RANGLIST and the TARGET_VIEW parameters. The NUMRANGE parameter is optional.</td>
</tr>
<tr>
<td>UNSHARE</td>
<td>Requests that the specified virtual storage no longer be used to access shared storage. When you issue the IARVSERV macro with UNSHARE, you must specify the RANGLIST parameter. The NUMRANGE, and RETAIN parameters are optional. Using the RETAIN parameter can allow the target area data to remain available to other programs that can access the target area.</td>
</tr>
<tr>
<td>CHANGEACCESS</td>
<td></td>
</tr>
<tr>
<td>RANGLIST=ranglist_addr</td>
<td>ranglist_addr: RS-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>NUMRANGE=numrange_addr</td>
<td>numrange_addr: RS-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>TARGET_VIEW=READONLY</td>
<td></td>
</tr>
<tr>
<td>TARGET_VIEW=SHAREDWRITE</td>
<td></td>
</tr>
<tr>
<td>TARGET_VIEW=UNIQUEWRITE</td>
<td></td>
</tr>
<tr>
<td>TARGET_VIEW=TARGETWRITE</td>
<td></td>
</tr>
<tr>
<td>TARGET_VIEW=LIKESOURCE</td>
<td></td>
</tr>
<tr>
<td>TARGET_VIEW=HIDDEN</td>
<td></td>
</tr>
<tr>
<td>COPYNOW</td>
<td></td>
</tr>
<tr>
<td>RETAIN=NO</td>
<td>Default: RETAIN=NO</td>
</tr>
<tr>
<td>RETAIN=YES</td>
<td></td>
</tr>
<tr>
<td>PLISTVER=IMPLIED_VERSION</td>
<td></td>
</tr>
<tr>
<td>PLISTVER=MAX</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>PLISTVER=plistver</td>
<td>plistver: 0</td>
</tr>
</tbody>
</table>

### Parameters

The SHARE, UNSHARE, and CHANGEACCESS parameters designate the services of the IARVSERV macro, and are mutually exclusive.

The parameters are explained as follows:

**SHARE**

Requests that the source be made shareable through the target to create a sharing group. When you issue the IARVSERV macro with SHARE, you must specify the RANGLIST and the TARGET_VIEW parameters. The NUMRANGE parameter is optional.

**UNSHARE**

Requests that the specified virtual storage no longer be used to access shared storage. When you issue the IARVSERV macro with UNSHARE, you must specify the RANGLIST parameter. The NUMRANGE, and RETAIN parameters are optional. Using the RETAIN parameter can allow the target area data to remain available to other programs that can access the target area.
CHANGEACCESS
Requests that the type of access to the specified virtual storage be changed.
When you issue the IARVSERV macro with CHANGEACCESS, you must specify the RANGLIST and the TARGET_VIEW parameters. The NUMRANGE parameter is optional.

\texttt{RANGLIST=ranglist\_addr}
Specifies the name (RS-type) or address (in register 2-12) of a required input fullword that contains the address of the range list. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 28 bytes long. A mapping of each entry is provided through the mapping macro IARVRL.

\texttt{NUMRANGE=numrange\_addr}
Specifies the name (RS-type) or address (in register 2-12) of an optional parameter that provides the number of entries in the supplied RANGLIST. Only authorized programs can specify more that 16 entries in the range list. If you do not specify NUMRANGE, the system assumes the range list contains only one entry.

\texttt{TARGET\_VIEW=READONLY}
\texttt{TARGET\_VIEW=SHAREDWRITE}
\texttt{TARGET\_VIEW=UNIQUEWRITE}
\texttt{TARGET\_VIEW=TARGETWRITE}
\texttt{TARGET\_VIEW=LIKESOURCE}
\texttt{TARGET\_VIEW=HIDDEN}
Specifies the way you want to share storage when used on storage not already part of a sharing group, or how you want to change or add storage access to the sharing group for storage already shared.

The keywords that are valid for TARGET_VIEW and their meanings follow:

\textbf{READONLY}
Specifies that the target can be used only to read shared data. Any attempt to alter shared data by writing into the target will cause a program check.

\textbf{SHAREDWRITE}
Specifies that the target can be used to read or modify shared data. When a program changes data in the target, the new data becomes visible among all those programs that have READONLY and SHAREDWRITE access to the source. Those programs with UNIQUEWRITE access to the source will not see the changed data.

\textbf{UNIQUEWRITE}
Specifies that the target can be used to read shared data and to retain a private copy of the shared data should the source or any target get altered. When another user of the target modifies the data, the page in the target containing the modified data becomes a private copy that is unique to that user (with UNIQUEWRITE) and not accessible to any other program.

\textbf{TARGETWRITE}
Specifies that the target can be used to read shared data and retain a private copy of the shared data if this view of the shared data is altered. When another user of the target area writes new data into the target area, any page in the target area containing the new data becomes a private copy that is unique and is not seen by to any other user. The page is no longer a member of any sharing group. The
original source data is unchanged. When a SHAREDWRITE view of the data gets altered, the TARGETWRITE view will see those changes. 

LIKESOURCE 
Specifies that the view type for the new target area is to be the same as the current view of the source. If the source is not currently shared, a copy of the source is made to the new target as if COPYNOW had been coded.

HIDDEN 
Specifies that the data in the target area will be inaccessible until the view type is changed to READONLY, SHAREDWRITE, UNIQUEWRITE, or TARGETWRITE. Any attempt to access a hidden target area will cause a program check.

,COPYNOW 
Specifies whether the target should get a copy of the source data when using UNIQUEWRITE or LIKESOURCE. You can use COPYNOW only when you specify TARGET_VIEW=UNIQUEWRITE or TARGET_VIEW=LIKESOURCE.

,RETAIN=YES,RETAIN=NO 
Specifies whether a copy of the shared data is to be retained in the target after the system finishes processing the UNSHARE request.

RETAIN=YES 
Specifies that the target view should retain a copy of the shared data. Using UNSHARE with RETAIN=YES requires the system to allocate new resources to back the target area.

RETAIN=NO 
Specifies that the contents of the target area are unpredictable. To ensure zeroes, the user should issue a PGSER RELEASE or DSPSERV RELEASE on the area after unsharing it. RETAIN=NO is the default.

Note: PGRLSE, PGSER RELEASE, PGSER FREE with RELEASE=Y, and PGFREE RELEASE=Y may ignore some or all of the pages in the input range and will not notify the caller if this was done.

Any pages in the input range that match any of the following conditions will be skipped, and processing continues with the next page in the range:
• Storage is not allocated or all pages in a segment have not yet been referenced.
• Page is in PSA, SQA or LSQA.
• Page is V=R. Effectively, it's fixed.
• Page is in BLDL, (E)PLPA, or (E)MLPA.
• Page has a page fix in progress or a nonzero FIX count.
• Pages with COMMIT in progress or with DISASSOCIATE in progress.

,PLISTVER=IMPLIED_VERSION,PLISTVER=MAX,PLISTVER=plistver 
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:
**IARVSERV macro**

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
  
  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, if you use the currently available parameters.

**To code**, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

**ABEND codes**

IARVSERV might abnormally terminate with the abend code X'6C5'. See [z/OS MVS System Codes](#) for an explanation and programmer response.

**Return and reason codes**

When the IARVSERV macro returns control to your program, GPR 15 contains the return code. If the return code is not 0, GPR 0 contains the reason code.

*Table 36. Return and Reason Codes for the IARVSERV Macro*

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | None                    | Meaning: The IARVSERV request completed successfully.  
  Action: None required. |
| 04                      | xx0101xx                | Meaning: IARVSERV SHARE completed successfully. The processor does not support SHARE for UNIQUEWRITE. A unique copy of the target was made by the system.  
  Action: None required. |
| 04                      | xx0102xx                | Meaning: IARVSERV SHARE completed successfully. However, the system found a condition that would lead to a storage requirement conflict for sharing with UNIQUEWRITE. For example, the source might be in non-pageable storage. A copy of the target was made by the system to avoid this conflict.  
  Action: None required. However, you might want to correct the storage conflict. |
### Table 36. Return and Reason Codes for the IARVSERV Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>04 xx0103xx</td>
<td></td>
<td><strong>Meaning</strong>: IARVSERV SHARE found that some source pages were not obtained using the GETMAIN or STORAGE macros, or the source and target keys do not match and the request is for a UNIQUEWRITE target view. If the corresponding target pages were obtained using the GETMAIN or STORAGE macro, then they have been made first reference. <strong>Action</strong>: This is not necessarily an error. If you think you should not get this reason code, check program to be sure GETMAIN or STORAGE is issued and storage is of the same storage key for all source and target storage prior to using IARVSERV.</td>
</tr>
<tr>
<td>04 xx0203xx</td>
<td></td>
<td><strong>Meaning</strong>: IARVSERV UNSHARE completed successfully. However, the system has overridden the RETAIN=NO option and kept a copy of the data in the target. <strong>Action</strong>: None required. However, you may want to correct your use of DIV.</td>
</tr>
<tr>
<td>04 xx0204xx</td>
<td></td>
<td><strong>Meaning</strong>: IARVSERV UNSHARE completed successfully. The system has overridden the RETAIN=YES option because the shared data is associated with a DIV object, and the target area is not the original window mapped to the DIV object. The data in the target is unpredictable. <strong>Action</strong>: None required.</td>
</tr>
<tr>
<td>04 xx0205xx</td>
<td></td>
<td><strong>Meaning</strong>: IARVSERV UNSHARE completed successfully. Some pages in the target area no longer belong to any sharing group. This could be due to a copy being created by UNIQUEWRITE, or a second invocation of UNSHARE on the same view. <strong>Action</strong>: None required.</td>
</tr>
<tr>
<td>04 xx0301xx</td>
<td></td>
<td><strong>Meaning</strong>: IARVSERV CHANGEACCESS completed successfully. The processor does not support CHANGEACCESS for UNIQUEWRITE, and a unique copy of the target page was made. <strong>Action</strong>: None required.</td>
</tr>
<tr>
<td>04 xx030Cxx</td>
<td></td>
<td><strong>Meaning</strong>: IARVSERV CHANGEACCESS completed successfully. The system processed a CHANGEACCESS request for UNIQUEWRITE or TARGETWRITE for non-shared pages as a SHAREDWRITE request. <strong>Action</strong>: None required.</td>
</tr>
</tbody>
</table>
### IARVSERV macro

#### Table 36. Return and Reason Codes for the IARVSERV Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08                      | xx0104xx                | **Meaning**: Environmental error. An unauthorized user attempted to share more pages than allowed by the installation (as defined through the installation exit IEFUSI).  
**Action**: Contact your system programmer to find out your installation limit and reduce the number of shared pages. |
| 08                      | xx0105xx                | **Meaning**: Environmental error. IARVSERV SHARE was requested with TARGETWRITE, but the SOP hardware feature was not available.  
**Action**: Contact your system programmer to find out when the SOP feature might become available. |
| 08                      | xx0305xx                | **Meaning**: Environmental error. IARVSERV CHANGEACCESS was requested with TARGETWRITE, but the SOP hardware feature was not available.  
**Action**: Contact your system programmer to find out when the SOP feature may become available. |
| 0C                      | xx010Axx                | **Meaning**: Environmental error. IARVSERV SHARE cannot complete the request because of a shortage of resources.  
**Action**: Retry the request one or more times to see if resources become available. Contact the system programmer to determine resources available to you. |
| 0C                      | xx013Cxx                | **Meaning**: System error. IARVSERV SHARE cannot complete the request because a required page is unavailable or lost.  
**Action**: Check the paging data set for possible I/O errors. Refer to X'028' abend description in [z/OS MVS System Codes](https://www.ibm.com/support/knowledgecenter/SYSXKR_2.1.0/com.ibm.mvs.doc/abend.html) for paging error advice. |
| 0C                      | xx020Bxx                | **Meaning**: System error. IARVSERV UNSHARE cannot complete the request because of a required page being unavailable or lost.  
**Action**: Check the logrec data set for possible I/O errors. Refer to X'028' abend description in [z/OS MVS System Codes](https://www.ibm.com/support/knowledgecenter/SYSXKR_2.1.0/com.ibm.mvs.doc/abend.html) for paging error advice. |
| 0C                      | xx030Bxx                | **Meaning**: System error. IARVSERV CHANGEACCESS cannot complete the request because of a required page being unavailable or lost.  
**Action**: Check the logrec data set for possible I/O errors. Refer to X'028' abend description in [z/OS MVS System Codes](https://www.ibm.com/support/knowledgecenter/SYSXKR_2.1.0/com.ibm.mvs.doc/abend.html) for paging error advice. |

#### Example 1

Issue a request to share eight pages as read-only, and use a register to specify the address of the range list.
Example 2

Issue UNSHARE for the pages in Example 1, and specify that the system is not to retain the shared data.

SERV2 IARVSERV UNSHARE,RANGLIST=(4),RETAIN=NO
* IARVRL

Example 3

Issue a request to share pages as read-only, and use an RS-type address to specify the location of the range list address.

SERV3 IARVSERV SHARE,RANGLIST=VRLPTR,TARGET_VIEW=READONLY
* VRLPTR DC A(MYVRL1)
MYVRL1 DS 7F
IARVRL

Example 4

Issue a request to share pages as target write.

SERV4 IARVSERV SHARE,RANGLIST=(5),TARGET_VIEW=TARGETWRITE
* IARVRL

Example 5

Issue a request to change access for hidden.

SERV5 IARVSERV CHANGEACCESS,RANGLIST=(5),TARGET_VIEW=HIDDEN
* IARVRL

IARVSERV—List form

Use the list form of the IARVSERV macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

The list form of the IARVSERV macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARVSERV.</td>
</tr>
<tr>
<td>IARVSERV</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARVSERV.</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td></td>
</tr>
</tbody>
</table>
IARVSERV macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,PLISTVER=MAX</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=plistver</td>
<td>plistver: 0</td>
</tr>
<tr>
<td>MF=(L,list addr)</td>
<td>list addr: symbol.</td>
</tr>
<tr>
<td>MF=(L,list addr,attr)</td>
<td>attr: 1- to 60-character input string.</td>
</tr>
<tr>
<td>MF=(L,list addr,0D)</td>
<td>Default: 0D</td>
</tr>
</tbody>
</table>

The parameters are explained under the standard form of the IARVSERV macro with the following exception:

MF=(L,list addr)
MF=(L,list addr,attr)
MF=(L,list addr,0D)

Specifies the list form of the IARVSERV macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

IARVSERV - Execute form

Use the execute form of the IARVSERV macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The execute form of the IARVSERV macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARVSERV.</td>
</tr>
<tr>
<td>IARVSERV</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARVSERV.</td>
</tr>
<tr>
<td>SHARE</td>
<td></td>
</tr>
<tr>
<td>UNSHARE</td>
<td></td>
</tr>
<tr>
<td>CHANGEACCESS</td>
<td></td>
</tr>
<tr>
<td>,RANGLIST=ranglist_addr</td>
<td>ranglist_addr: RS-type address, or address in register (2) - (12).</td>
</tr>
</tbody>
</table>
IARVSERV macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,NUMRANGE=\texttt{numrange_addr}</td>
<td>numrange_addr: RS-type address, or address in register (2) - (12). Default: 1 range</td>
</tr>
</tbody>
</table>

| ,TARGET\_VIEW=\texttt{READONLY} | |
| ,TARGET\_VIEW=\texttt{SHAREDWRITE} | |
| ,TARGET\_VIEW=\texttt{UNIQUEWRITE} | |
| ,TARGET\_VIEW=\texttt{TARGETWRITE} | |
| ,TARGET\_VIEW=\texttt{LIKESOURCE} | |
| ,TARGET\_VIEW=\texttt{HIDDEN} | |

| ,COPYNOW | |

| ,RETAIN=\texttt{NO} | Default: RETAIN=\texttt{NO} |
| ,RETAIN=\texttt{YES} | |

| ,PLISTVER=\texttt{IMPLIED\_VERSION} | Default: IMPLIED\_VERSION |
| ,PLISTVER=\texttt{MAX} | |
| ,PLISTVER=\texttt{plistver} | plistver: 0 |

| ,MF=(\texttt{E,\texttt{list\_addr}}) | \texttt{list\_addr}: RX-type address or address in register (2) - (12). |
| ,MF=(\texttt{E,\texttt{list\_addr,COMPLETE}}) | Default: COMPLETE |
| ,MF=(\texttt{E,\texttt{list\_addr,NOCHECK}}) | |

The parameters are explained under the standard form of the IARVSERV macro with the following exception:

\texttt{,MF=(\texttt{E,\texttt{list\_addr}})}
\texttt{,MF=(\texttt{E,\texttt{list\_addr,COMPLETE}})}
\texttt{,MF=(\texttt{E,\texttt{list\_addr,NOCHECK}})}

Specifies the execute form of the IARVSERV macro.

\texttt{list\_addr} specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.
Chapter 29. IARV64 — 64–bit virtual storage allocation

Description

The IARV64 macro allows a program to use the full range of virtual storage in an address space that is supported by 64-bit addresses. The macro creates and frees storage areas above the two gigabyte address and manages the physical frames behind the storage. Each storage area is a multiple of one megabyte in size and begins on a megabyte boundary. You can think of the IARV64 macro as the GETMAIN/FREEMAIN, PGSER or STORAGE macro for virtual storage above the two gigabyte address.

The two gigabyte address in the address space is marked by a virtual line called the bar. The bar separates storage below the two gigabyte address, called below the bar, from storage above the two gigabyte address, called above the bar. The area above the bar is intended to be used for data only, not for executing programs. Programs use the IARV64 macro to obtain storage above the bar in “chunks” of virtual storage called memory objects. Your installation can set a limit on the use of the address space above the bar for a single address space. The limit is called the MEMLIMIT.

When you create a nonshared, non-2 GB memory object, you can specify a guard area (not accessible) and a usable area. Subsequently, you can create alternate guard areas or change all or some of a guard area into an accessible area, or vice versa.

The following services are provided:

GETSTOR
Create a private memory object (in "REQUEST=GETSTOR option of IARV64" on page 343)

PAGEFIX
Fix physical pages within one or more nonshared memory objects. (in "REQUEST=PAGEFIX option of IARV64" on page 359)

PAGEUNFIX
Unfix physical pages within one or more nonshared memory objects. (in "REQUEST=PAGEUNFIX option of IARV64" on page 366)

PAGEOUT
Notify the system that data within physical pages of one or more memory objects will not be used in the near future. (in "REQUEST=PAGEOUT option of IARV64" on page 372)

PAGEIN
Notify the system that data within physical pages of one or more memory objects are needed in the near future. (in "REQUEST=PAGEIN option of IARV64" on page 377)

DISCARDDATA
Discard data within physical pages of one or more memory objects. (in "REQUEST=DISCARDDATA option of IARV64" on page 382)

CHANGEGUARD
Request that a specified range in a nonshared, non-2GB memory object be
IARV64 macro

changed from guard area to usable area or vice versa. (in “REQUEST=CHANGEGUARD option of IARV64” on page 388)

PROTECT
Request that data within one or more memory objects be made read-only. (in “REQUEST=PROTECT option of IARV64” on page 396)

UNPROTECT
Request that data within one or more memory objects be made modifiable. (in “REQUEST=UNPROTECT option of IARV64” on page 402)

LIST
Request a list of memory objects. (in “REQUEST=LIST option of IARV64” on page 409)

DETACH
Free one or more memory objects. For a nonshared memory object, the object is freed. For a shared memory object, the object is freed only when the last shared user of that memory object issues the DETACH (this includes a DETACH corresponding to the system attachment formed when the object was created through GETSHARED). (in “REQUEST=DETACH option of IARV64” on page 420)

GETSHARED
Create a memory object that can be shared across multiple address spaces. (in “REQUEST=GETSHARED option of IARV64” on page 429)

SHAREMEMOBJ
Request that the specified address space be given access to one or more specified shared memory objects. (in “REQUEST=SHAREMEMOBJ option of IARV64” on page 435)

CHANGEACCESS
Request that a view type for segments within the specified shared memory objects be changed. (in “REQUEST=CHANGEACCESS option of IARV64” on page 441)

GETCOMMON
Create a 64-bit common memory object. (in “REQUEST=GETCOMMON option of IARV64” on page 447)

COUNTPAGES
Count the number of 4K pages currently in use in real storage, on auxiliary storage, and in both real storage and on auxiliary storage to back the input high virtual storage ranges. (in “REQUEST=COUNTPAGES option of IARV64” on page 459)

For guidance information about the use of 64-bit virtual storage allocation, see z/OS MVS Programming: Extended Addressability Guide.

After the separate descriptions of each individual Request are the following topics which apply to all of the Requests:

• The abend codes in “ABEND codes” on page 465,
• The return and reason codes in “Return and reason codes” on page 466, and
• Examples of using IARV64 in “Example” on page 468

Note: The examples apply to REQUEST=GETSTOR, PAGEFIX, PAGEUNFIX, and DETACH.

Facts associated with these services:
A segment represents one megabyte of virtual storage starting on a megabyte boundary.

The storage returned by the GETSTOR, GETSHARED, or GETCOMMON services is called a memory object.

The storage returned by GETSHARED is referred to as a "shared memory object".

The storage returned by GETSTOR is referred to as a "private memory object" or a "system memory object".

The storage returned by GETCOMMON is referred to as a "common memory object".

The limit of storage per address space allowed to be used above the bar is called the MEMLIMIT. This is similar to the region parameter for storage below the bar. The following categories of storage do not count against the MEMLIMIT:
- The guard area in a memory object.
- The storage created by IARV64 GETSTOR with LOCALSYSAREA=YES.
- Shared memory objects, such as storage created by IARV64 GETSHARED.

REQUEST=GETSTOR option of IARV64

REQUEST=GETSTOR allows you to create a memory object. To avoid an abend for exceeding MEMLIMIT, specify the COND=YES parameter.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or with PSW key 0-7.</td>
</tr>
</tbody>
</table>

To use the PAGEFRAMESIZE parameter, a caller can be in problem state with either one of the following authorizations:
- APF-authorized
- Authorized for read to IARRSM.LRGPAGES

<table>
<thead>
<tr>
<th>Dispatchable unit mode</th>
<th>Task or SRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>Note:</td>
<td>The problem state caller running in PSW key 8-15 can use GETSTOR/DETACH only when the primary address space is the home address space.</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>• Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td></td>
<td>• Disabled for 64-bit common memory objects allocated</td>
</tr>
<tr>
<td></td>
<td>with TYPE=DREF.</td>
</tr>
<tr>
<td></td>
<td>• Disabled for TYPE=PAGEABLE and the storage is in the first reference state.</td>
</tr>
<tr>
<td>Locks:</td>
<td>A local lock may be held, subject to the following limitation:</td>
</tr>
</tbody>
</table>

When a local lock is held for a request (GETSTOR, SHAREMEMOBJ, DETACH, CHANGEGUARD, or DISCARDDATA) the lock must be for the address space specified or set as the default by the input ALETVALUE.

Control parameters: Control parameters must be in the primary address space and can reside both below and above the bar.
IARV64 macro

Programming requirements
  None

Restrictions
  This macro supports multiple versions. Some keywords are unique to certain
  versions. See PLISTVER parameter description.

Input register information
  Before issuing the IARV64 macro, the caller does not have to place any information
  into any register unless using it in register notation for a particular parameter, or
  using it as a base register.

Output register information
  When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

  When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

  Some callers depend on register contents remaining the same before and after
  issuing a service. If the system changes the contents of registers on which the caller
  depends, the caller must save them before issuing the service, and restore them
  after the system returns control.

Performance implications
  None

Syntax
  The REQUEST=GETSTOR option of the IARV64 macro is written as follows:

  Syntax          Description
  ---------------------------------------------------
  name            name: symbol. Begin name in column 1.
  b               One or more blanks must precede IARV64.
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=GETSTOR</td>
<td></td>
</tr>
<tr>
<td>_COND=NO</td>
<td>Default: COND=NO</td>
</tr>
<tr>
<td>_COND=YES</td>
<td></td>
</tr>
<tr>
<td>_LOCALSYSAREA=NO</td>
<td>Default: LOCALSYSAREA=NO</td>
</tr>
<tr>
<td>_LOCALSYSAREA=YES</td>
<td></td>
</tr>
<tr>
<td>_SEGMENTS='segments'</td>
<td>segments: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_TYPE=DREF</td>
<td></td>
</tr>
<tr>
<td>_TYPE=PAGEABLE</td>
<td>Default: TYPE=PAGEABLE when one of the following is specified:</td>
</tr>
<tr>
<td></td>
<td>• PAGEFRAMESIZE=PAGEABLE1MEG</td>
</tr>
<tr>
<td></td>
<td>• PAGEFRAMESIZE=4K</td>
</tr>
<tr>
<td></td>
<td>• PAGEFRAMESIZE=MAX and the memory object is backed with 4 KB-page frames.</td>
</tr>
<tr>
<td></td>
<td>When PAGEFRAMESIZE=DREF1MEG is specified, the default value is TYPE=DREF.</td>
</tr>
<tr>
<td></td>
<td>If PAGEFRAMESIZE=1MEG or PAGEFRAMESIZE=MAX is specified and the memory</td>
</tr>
<tr>
<td></td>
<td>object is backed with 1 MB-page frames, the 1 MB-pages backing this memory</td>
</tr>
<tr>
<td></td>
<td>object are fixed.</td>
</tr>
<tr>
<td>_PAGEFRAMESIZE=4K</td>
<td>Default: PAGEFRAMESIZE=4K</td>
</tr>
<tr>
<td>_PAGEFRAMESIZE=1MEG</td>
<td></td>
</tr>
<tr>
<td>_PAGEFRAMESIZE=MAX</td>
<td></td>
</tr>
<tr>
<td>_PAGEFRAMESIZE=PAGEABLE1MEG</td>
<td></td>
</tr>
<tr>
<td>_PAGEFRAMESIZE=DREF1MEG</td>
<td></td>
</tr>
<tr>
<td>_UNITS=units</td>
<td>units: Size of the memory object, which is the number of units specified</td>
</tr>
<tr>
<td></td>
<td>by UNITSIZE.</td>
</tr>
<tr>
<td>_UNITSIZE=1M</td>
<td>Specifies a 1 MB unit size.</td>
</tr>
<tr>
<td>_PAGEFRAMESIZE=4K</td>
<td>1M</td>
</tr>
<tr>
<td>_UNITSIZE=2G</td>
<td>Specifies a 2 GB unit size.</td>
</tr>
<tr>
<td>_PAGEFRAMESIZE=4K</td>
<td>1M</td>
</tr>
<tr>
<td>_TYPE=FIXED</td>
<td></td>
</tr>
<tr>
<td>_TYPE=DREF</td>
<td></td>
</tr>
<tr>
<td>_TYPE=PAGEABLE</td>
<td></td>
</tr>
</tbody>
</table>
### IARV64 Macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.KEY=key</td>
<td>key: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>.KEY=CALLERKEY</td>
<td>Default: KEY=CALLERKEY</td>
</tr>
<tr>
<td>.FPROT=YES</td>
<td>Default: FPROT=YES</td>
</tr>
<tr>
<td>.FPROT=NO</td>
<td></td>
</tr>
<tr>
<td>.MEMLIMIT=NO</td>
<td>Default: MEMLIMIT=YES</td>
</tr>
<tr>
<td>.MEMLIMIT=YES</td>
<td></td>
</tr>
<tr>
<td>.SVCDUMPRGN=YES</td>
<td>Default: SVCDUMPRGN=YES</td>
</tr>
<tr>
<td>.SVCDUMPRGN=NO</td>
<td></td>
</tr>
<tr>
<td>.DUMP=LIKERGN</td>
<td>Default: DUMP=LIKERGN</td>
</tr>
<tr>
<td>.DUMPPRIORITY=99</td>
<td>Default: DUMPPRIORITY=99</td>
</tr>
<tr>
<td>.DUMPPRIORITY=dumppriority</td>
<td></td>
</tr>
<tr>
<td>.DUMP=LIKELSQA</td>
<td></td>
</tr>
<tr>
<td>.DUMP=NO</td>
<td></td>
</tr>
<tr>
<td>.DUMP=BYPRIORITY</td>
<td></td>
</tr>
<tr>
<td>.OPTIONVALUE=option</td>
<td>option: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>.CONTROL=UNAUTH</td>
<td>Default: CONTROL=UNAUTH</td>
</tr>
<tr>
<td>.CONTROL=AUTH</td>
<td></td>
</tr>
<tr>
<td>.MOTKNSOURCE=USER</td>
<td>Default: MOTKNSOURCE=USER</td>
</tr>
<tr>
<td>.MOTKN=motkn</td>
<td>Default: MOTKN</td>
</tr>
<tr>
<td>.MOTKNCREATOR=USER</td>
<td>Default: MOTKNCREATOR=USER</td>
</tr>
<tr>
<td>.MOTKNCREATOR=SYSTEM</td>
<td></td>
</tr>
<tr>
<td>.USERTKN=NO USERTKN</td>
<td>Default: USERTKN=NO USERTKN</td>
</tr>
<tr>
<td>.USERTKN=usertkn</td>
<td>usertkn: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>.MOTKNSOURCE=SYSTEM</td>
<td></td>
</tr>
<tr>
<td>.OUTMOTKN=outmotkn</td>
<td>Outmotkn: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>.USERTKN=usertkn</td>
<td>usertkn: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>.USERTKN=NO USERTKN</td>
<td>Default: USERTKN=NO USERTKN</td>
</tr>
<tr>
<td>.GUARDSIZE=guardsize</td>
<td>guardsize: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>.GUARDSIZE=0</td>
<td>Default: GUARDSIZE=0</td>
</tr>
<tr>
<td>.GUARDSIZE64=guardsize64</td>
<td>guardsize64: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>.GUARDSIZE64=0</td>
<td>Default: GUARDSIZE64=0</td>
</tr>
</tbody>
</table>
Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUARDLOC=LOW</td>
<td>Default: GUARDLOC=LOW</td>
</tr>
<tr>
<td>GUARDLOC=HIGH</td>
<td></td>
</tr>
<tr>
<td>TTOKEN=ttoken</td>
<td>ttoken: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>TTOKEN=NO_TTOKEN</td>
<td>Default: TTOKEN=NO_TTOKEN</td>
</tr>
<tr>
<td>ALETVALUE=aletvale</td>
<td>aletvalue: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>ALETVALUE=0</td>
<td>Default: ALETVALUE=0</td>
</tr>
<tr>
<td>ORIGIN=origin</td>
<td>origin: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>DETACHFIXED=NO</td>
<td>Default: DETACHFIXED=NO</td>
</tr>
<tr>
<td>DETACHFIXED=YES</td>
<td></td>
</tr>
<tr>
<td>RETCODE=rcode</td>
<td>recode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>PLISTVER=0, 1, 2, 3, 4, 5</td>
<td></td>
</tr>
<tr>
<td>MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>MF=(L,list addr)</td>
<td>list addr: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td>MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=GETSTOR
A required parameter. REQUEST=GETSTOR creates a private memory object if LOCALSYSAREA=YES is not specified. If LOCALSYSAREA=YES is specified, then a system memory object is returned. The storage obtained in the system area using the LOCALSYSAREA keyword will not be copied during Fork

Chapter 29. IARV64 — 64–bit virtual storage allocation
processing. The use of local system area storage does not preclude checkpoint from succeeding. At completion, the memory object is created in the address space you indicate.

\textbf{COND=NO}\textbf{,COND=YES}\textbf{,COND=NO}\textbf{,COND=YES}

An optional input parameter that specifies whether the request is unconditional or conditional. If you code COND=YES and there is insufficient storage to satisfy the request, instead of the request being abnormally ended, the request will complete, but a return code will be set to indicate that the request could not be completed successfully. In all cases, the request will be abnormally ended for invalid requests, including violation of environmental restrictions. The default is COND=NO.

\textbf{COND=NO}\textbf{,COND=NO}\textbf{,COND=NO}\textbf{,COND=NO}

The request is unconditional. The request is abnormally ended when the request cannot be satisfied.

\textbf{COND=NO}\textbf{,COND=NO}\textbf{,COND=NO}\textbf{,COND=NO}

The request is conditional. The request is not abnormally ended for resource unavailability.

\textbf{LOCALSYSAREA=NO}\textbf{,LOCALSYSAREA=NO}\textbf{,LOCALSYSAREA=NO}\textbf{,LOCALSYSAREA=NO}

An optional input parameter that specifies whether this is an explicit allocation request for 64-bit virtual storage in local system area. The localsysarea parameter can be used only by callers running in supervisor state or with a PSW key 0-7. The default is LOCALSYSAREA= NO.

\textbf{LOCALSYSAREA=NO}\textbf{,LOCALSYSAREA=NO}\textbf{,LOCALSYSAREA=NO}\textbf{,LOCALSYSAREA=NO}

The request will not be satisfied from the local system area.

\textbf{LOCALSYSAREA=YES}\textbf{,LOCALSYSAREA=YES}\textbf{,LOCALSYSAREA=YES}\textbf{,LOCALSYSAREA=YES}

The request is to be satisfied from the local system area. The storage obtained with this keyword will not be copied during Fork processing. The use of local system area storage does not preclude checkpoint from succeeding.

\textbf{KEY=key}\textbf{,KEY=key}\textbf{,KEY=key}\textbf{,KEY=key}

An optional input parameter that specifies the storage key to be assigned to the memory object. The key must be in bits 0-3 of the specified byte. Bits 4-7 are ignored. The KEY parameter can be used only by callers running in supervisor state or with a PSW key 0-7; with the following exception: a PSW key 8 caller can specify a storage key of the memory object to be key 9.

If the key is not specified, the storage key of the memory object is the same as the caller's PSW key. The default is CALLERKEY.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

\textbf{FPROT=YES}\textbf{,FPROT=YES}\textbf{,FPROT=YES}\textbf{,FPROT=YES}

An optional input parameter that specifies whether the memory object should be fetch-protected. The default is FPROT=YES.

\textbf{FPROT=YES}\textbf{,FPROT=YES}\textbf{,FPROT=YES}\textbf{,FPROT=YES}

The entire memory object is fetch-protected. A program must have a PSW key that matches the storage key of the memory object (or have PSW key 0) to reference data in the memory object.
The memory object is not fetch-protected.

An optional input parameter that specified whether the allocation of the 64-bit memory object is to count towards the address space MEMLIMIT. The default is MEMLIMIT=YES.

The 64-bit private memory object contributes towards the address space MEMLIMIT.

The 64-bit private memory object is not counted against the address space MEMLIMIT. MEMLIMIT=NO can only be specified by authorized callers in supervisor state or key 0-7.

SEGMENTS=segments

SEGMENTS and UNITS are mutually exclusive keys. This set is required; only one key can be specified.

A required input parameter that specifies the size of the memory object requested, in megabytes. This must be a nonzero value. The amount of storage requested that is not in the guard state is charged against the MEMLIMIT for the address space where the memory object is to be created.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

An optional input parameter that specifies the size of the page frames to back the virtual storage mapped by the allocated memory object.

The memory object is backed by 4 KB-page frames, if available. This is the default value.

The memory object is backed by 1 MB-page frames, if available.

The memory object is backed by the largest page frame size that is supported and available. Otherwise, the object is backed by 4 KB-page frames. 1 MB-page frames are backed at allocation and cannot be paged out to AUX. 4 KB-page frames are backed at first reference and can be paged out to AUX as long as TYPE=DREF is not specified.

The memory object is backed by pageable 1 MB-page frames at first reference, unless none are available. If none are available, the object is backed by 4 KB-page frames.

The memory object is backed by Dref 1 MB-page frames at first reference, unless none are available. If none are available, the object is backed by 4 KB-page frames.
An optional input parameter that specifies the type of storage that is requested. The default value is TYPE=PAGEABLE when one of the following parameters is specified:
- PAGEFRAMESIZE=4K
- PAGEFRAMESIZE=PAGEABLE1MEG
- PAGEFRAMESIZE=MAX and the memory object is backed with 4 KB-page frames.

The default value is TYPE=DREF when PAGEFRAMESIZE=DREF1MEG is specified. If PAGEFRAMESIZE=1MEG or PAGEFRAMESIZE=MAX is specified and the memory object is backed with 1 MB-page frames, the 1 MB-pages backing this memory object are fixed.

Note:
1. When the memory object is backed by 4 KB-page frames, the 4 KB-pages backing this memory object are pageable if TYPE=DREF is not specified, or are fixed if TYPE=DREF is specified. The 4 KB-pages are backed at first reference and can only be paged out to AUX if TYPE=DREF is not specified.
2. When the memory object is backed by 1 MB-page frames as a result of PAGEFRAMESIZE=PAGEABLE1MEG or PAGEFRAMESIZE=DREF1MEG being specified, the 1 MB-pages backing this memory object are pageable if PAGEABLE1MEG is specified or fixed if DREF1MEG is specified. Pageable 1 MB-pages are backed at first reference and can be paged out to AUX. DREF 1 MB-pages are backed at first reference and are fixed—they cannot be paged out to AUX.
3. When the memory object is backed by 1 MB-page frames because PAGEFRAMESIZE=1MEG or PAGEFRAMESIZE=MAX has been specified, the 1 MB-pages backing this memory object are fixed. Pages are backed at allocation time and cannot be paged out to AUX.

A required input parameter that specifies the size of the memory object as a number of units specified by the UNITSIZE parameter. This must be a nonzero value. The amount of storage requested that is not in the guard state is counted towards the MEMLIMIT for the address space where the memory object will be created. UNITSIZE belongs to a set of mutually exclusive keys. This set is required; only one key can be specified.

A required input parameter that specifies the type of requested storage.

Pages backing this memory object are pageable. Pages are backed at first reference and can be paged out to auxiliary storage. Virtual address ranges within the memory object can be explicitly fixed after allocation by using the IARV64 REQUEST=PAGEFIX request. TYPE=PAGEABLE is not valid with PAGEFRAMESIZE=2G.
, TYPE=DREF
Pages are backed in real memory at first reference, unless DREF storage is not available, in which case the program is ABENDed. Once backed, pages belonging to memory objects of TYPE=DREF remain in real storage and are never paged out to auxiliary storage. The memory object can be referenced while running disabled. The DREF attribute applies to the entire memory object. TYPE=DREF is not valid with PAGEFRAMESIZE=2G.

, TYPE=FIXED
Pages are backed in real storage immediately, unless fixed storage is not immediately available, in which case the request fails. Pages belonging to memory objects of TYPE=FIXED remain in real storage and are never paged out to auxiliary storage. The memory object can be referenced while running disabled. The FIXED attribute applies to the entire memory object when it is allocated. TYPE=FIXED is not valid with PAGEFRAMESIZE=4K.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

, UNITSIZE=1M
, UNITSIZE=2G
A required input parameter that specifies the size for the UNITS parameter: either 1M or 2 GB.

, UNITSIZE=1M
Specifies that the memory object is in one-megabyte (1 MB) units. For example, a request for UNITS=3 with UNITSIZE=1M is a request for three megabytes of virtual storage starting on a 1 MB boundary. When UNITSIZE=1M is specified, one of the following PAGEFRAMESIZE values must also be specified:

PAGEFRAMESIZE=4K
PAGEFRAMESIZE=1M
A required input parameter that specifies the size of the page frames used to back the virtual storage mapped by the allocated memory object.

PAGEFRAMESIZE=4K
Specifies to back the memory object with 4 KB-page frames of the specified TYPE, when TYPE=PAGEABLE or TYPE=DREF is requested. TYPE=FIXED is not supported.

PAGEFRAMESIZE=1M
Specifies to back the memory object by one-megabyte (1 MB) page frames of the specified TYPE. If 1 MB-page frames are not supported or not available, the system attempts to back the memory object at a smaller page frame size of the specified TYPE, when TYPE=PAGEABLE or TYPE=DREF is requested. A TYPE=FIXED request fails if there are no available pages in the requested PAGEFRAMESIZE.

, UNITSIZE=2G
Specifies that the memory object is in two-gigabyte (2G) units. For example, a request for UNITS=3 with UNITSIZE=2G is a request for six gigabytes of virtual storage starting on a 2 GB boundary.
IARV64 macro

**PAGEFRAMESIZE=4K|1M|2G**
A required input parameter that specifies the size of the page frames that back the virtual storage mapped by the allocated memory object.

**PAGEFRAMESIZE=4K**
Specifies to back the memory object by 4 KB-page frames of the specified TYPE, when TYPE=PAGEABLE or TYPE=DREF is requested. TYPE=FIXED is not supported with this value.

**PAGEFRAMESIZE=1M**
Specifies to back the memory object by one-megabyte (1 MB) page frames of the specified TYPE. If 1 MB-page frames are not supported or not available when TYPE=PAGEABLE or TYPE=DREF is requested, the system attempts to back the memory object using a smaller page frame size of the specified TYPE. A TYPE=FIXED request fails if there are no available pages in the requested PAGEFRAMESIZE.

**PAGEFRAMESIZE=2G**
Specifies to back the memory object by two-gigabyte (2 GB) FIXED page frames. PAGEFRAMESIZE=2G is valid only when TYPE=FIXED is specified. If 2 GB page frames are not supported or not available, the request fails.

**,SVCDUMPRGN=YES**
**,SVCDUMPRGN=NO**

**SVCDUMPRGN and DUMP** are mutually exclusive keys. This set is optional; only one key may be specified.

An optional input parameter that specifies whether the memory object should be included in an SVC dump when region is requested. The default is SVCDUMPRGN=YES for TYPE=PAGEABLE. If neither the SVCDUMPRGN keyword nor the DUMP keyword is specified the defaults that apply are as described under the defaults for the DUMP keyword.

**,SVCDUMPRGN=YES**
The memory object should be included in an SVC dump when RGN is specified on SDATA. This is equivalent to DUMP=LIKERGN.

**,SVCDUMPRGN=NO**
The memory object should not be included in an SVC dump when RGN is specified on SDATA.

**,DUMP=LIKERGN**
**,DUMP=LIKELSQA**
**,DUMP=NO**
**,DUMP=BYOPTIONVALUE**

**DUMP and SVCDUMPRGN** are mutually exclusive keys. This set is optional; only one key may be specified.

An optional input parameter that specifies whether the 64-bit private memory object will be included in an SVC dump when RGN or LSQA is specified on SDATA. When TYPE=PAGEABLE is specified on IARV64 GETSTOR the default is DUMP=LIKERGN. When TYPE=DREF is specified on IARV64 GETSTOR the default is DUMP=LIKELSQA. For memory objects backed with large pages the default is DUMP=NO.

**,DUMP=LIKERGN**
The 64-bit private memory object is included in an SVC dump when RGN is specified on SDATA.
**IARV64 macro**

\[,DUMPPRIORITY=99\]

\[,DUMPPRIORITY=dumppriority\]

An optional input parameter that specifies the dump priority of the memory object. This must be a non-zero value in the range of 1 to 99, with 1 being the highest priority and 99 being the lowest. The default is DUMPPRIORITY=99.

\[,DUMPPRIORITY=99\]

The dump priority of the memory object is 99 which is the lowest priority.

\[,DUMPPRIORITY=dumppriority\]

This parameter is the name (RS-type), or address in register (2)-(12), of an optional byte input that specifies the dump priority of the memory object. This must be a non-zero value in the range of 1 to 99, with 1 being the highest priority and 99 being the lowest.

\[,DUMP=LIKELSQA\]

The 64-bit private memory object is included in an SVC dump when LSQA is specified on SDATA.

\[,DUMP=NO\]

The 64-bit private memory object is not included in an SVC dump when either RGN or LSQA is specified on SDATA.

\[DUMP=BYOPTIONVALUE\]

The 64-bit private memory object is dumped according to the option specified by the OPTIONVALUE keyword.

\[,OPTIONVALUE=optionvalue\]

This parameter is the name of a required one-byte integer input that contains one of the dump option values as specified by the bit constants.

\[,CONTROL=UNAUTH\]

\[,CONTROL=AUTH\]

An optional input parameter that specifies when the memory object should be eligible for the certain other services.

This is a permanent attribute of the memory object and cannot be altered by other services. The default is CONTROL=UNAUTH.

\[,CONTROL=UNAUTH\]

The memory object can be freed by an unauthorized caller that owns the memory object. The memory object is NOT eligible for PAGEFIX.

\[,CONTROL=AUTH\]

The memory object can be freed only by an authorized caller. The memory object is eligible for PAGEFIX and PAGEUNFIX (note that PAGEFIX and PAGEUNFIX still require an authorized caller). AUTH can be used only by callers running in supervisor state or with PSW key 0-7.

\[MOTKNSOURCE=USER\]

\[MOTKNSOURCE=SYSTEM\]

An optional input parameter that indicates who provided (or will provide) the memory object token.

\[MOTKNSOURCE=USER\]

The user provides the memory object token.

The following is a set of mutually exclusive keys. This set is optional; only one key may be specified.
IARV64 macro

MOTKN=motkn

This parameter belongs to a set of mutually exclusive keys. It is the name of an optional doubleword integer input that identifies the user token to be associated with the memory object. This can be used on a later DETACH request to free all memory objects associated with this value.

- To request a system-generated token, use:
  IARV64 REQUEST(GETCOMMON) MOTKNSOURCE(SYSTEM) OUTMOTKN(mytoken)
- Use the returned token on subsequent IARV64 GETCOMMON requests, in order to associate other memory objects with the same token:
  IARV64 REQUEST(GETCOMMON) MOTKNSOURCE(USER) MOTKN(mytoken)

To avoid inadvertent collisions in the values specified, the left word (bits 0-31) of the user token must be binary zeros for a problem state program. The system enforces this requirement. The right word (bits 32-63) should represent the virtual address of some storage related to the caller, which could be a control block address, an entry point address, and so on, which is used as an application choice.

The convention for supervisor state program is that the left word (bits 0-31) should represent an address of some storage related to the caller. The system enforces the rule that the left word is nonzero for supervisor state callers. The format for the right word (bits 32-63) is a choice left to the caller.

If you specify no user token, the default is that no user token is supplied to associate this memory object with others.

MOTKNCREATOR=USER
MOTKNCREATOR=SYSTEM

This parameter is an optional input parameter that indicates who created the memory object token.

- Motkncreator=USER
  The memory object token is user-created.
- Motkncreator=SYSTEM
  The memory object token is system-created.

USERTKN=usertkn

This parameter belongs to a set of mutually exclusive keys. It is the name of an optional doubleword integer input that is a synonym for MOTKN. You can use either USERTKN or MOTKN interchangeably.

MOTKNSOURCE=SYSTEM

The system provides the memory object token.

OUTMOTKN=xoutmotkn

This parameter is the name of a required doubleword integer output that identifies the user token to be associated with the memory object to be created by the system.

USERTKN=usertkn
USERTKN=NO_USERTKN

An optional input parameter that identifies the user token to be associated
with the memory object. This can be used on a later DETACH request to free
all memory objects associated with this value.

To avoid inadvertent collisions in the values specified, the left word (bits 0-31)
of the user token must be binary zeros for a problem state program. The
system enforces this requirement. The right word (bits 32-63) should represent
the virtual address of some storage related to the caller, which could be a
control block address, an entry point address, and so on, which is used as an
application choice.

The convention for supervisor state program is that the left word (bits 0-31)
should represent an address of some storage related to the caller. The system
enforces the rule that the left word is nonzero for supervisor state callers. The
format for the right word (bits 32-63) is a choice left to the caller.

If you specify NO_USERTKN, the default is that no user token is supplied to
associate this memory object with others. The default is NO_USERTKN.

To code: Specify the RS-type address, or address in register (2)-(12), of a
doubleword field.

\texttt{GUARDSIZE=guardsize}
\texttt{GUARDSIZE=0}

GUARDSIZE and GUARDSIZE64 are mutually exclusive keys. This set is
optional; only one key may be specified. A fullword integer input parameter
that indicates the number of megabytes of guard area to be created at the high
or low end of the memory object. Guard areas cannot be referenced and when
referenced will cause a program check. Guard area does not count against the
MEMLIMIT. A guard area can be reduced through CHANGEGUARD
CONVERT=FROMGUARD.

GUARDSIZE must not be larger than the size of the memory object. The
default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a
fullword field.

\texttt{GUARDSIZE64=guardsize64}
\texttt{GUARDSIZE64=0}

GUARDSIZE64 belongs to a set of mutually exclusive keys. This set is optional;
only one key may be specified. A doubleword integer input parameter that
indicates the number of megabytes of guard area to be created at the high or
low end of the memory object. Guard areas cannot be referenced and when
referenced will cause a program check. Guard area does not count against the
MEMLIMIT. A guard area can be reduced through CHANGEGUARD
CONVERT=FROMGUARD.

GUARDSIZE64 must not be larger than the size of the memory object. The
default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a
doubleword field.

\texttt{GUARDLOC=LOW}
\texttt{GUARDLOC=HIGH}

An optional input parameter that specifies whether the guard location is at the
low virtual end of the memory object or the high virtual end. The default is
GUARDLOC=LOW.
The guard areas are created starting from the origin of the memory object, that is, from the low virtual end.

The guard areas are created at the end of the memory object, that is, at the high virtual end.

An optional input parameter that identifies the task to assume ownership of the memory object. The TTOKEN is returned by the TCBTOKEN macro.

If TTOKEN is specified, the task identified by the TTOKEN becomes the owner of the memory object. If TTOKEN is not specified, the currently dispatched task becomes the owner of the memory object. The task identified by the TTOKEN must be in the address space specified or defaulted by the ALETVALUE keyword.

The TTOKEN parameter must be used by a caller that is an SRB.

When the TTOKEN parameter is used by a problem state program with PSW key 8 - 15, the target task must represent the calling task OR the jobstep task for the calling task OR the mother task. A caller cannot assign ownership to a task above the jobstep task.

A memory object will be freed when its owning task terminates.

If the TTOKEN parameter is not specified, and the caller is a task (rather than an SRB), the currently dispatched task will become the owner of the memory object. An SRB will be abnormally ended if the TTOKEN parameter does not specify a valid TTOKEN. The default is NO_TTOKEN.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a 16 character field.

An optional input parameter that indicates the ALET of the address space in which the memory object is to be created.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a fullword field.

A required output parameter that contains the lowest address of the memory object. Note that when GUARDLOC=LOW is specified, the lowest address will point to a guard area which will cause an ABEND if referenced. For GUARDLOC=LOW the first usable area is the origin plus the size of the guard area.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

An optional input parameter that specifies whether the memory object can be detached when it contains fixed pages at the time of the DETACH request. The default value for DETACHFIXED is NO.
DETACHFIXED=NO
The memory object will not be detached if it has any fixed pages when it is
being detached.

DETACHFIXED=YES
The memory object will be detached even if some or all the pages of
that memory object are fixed.

,RETCODE=retcode
An optional output parameter into which the return code is to be copied from
GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=rsncode
An optional output parameter into which the reason code is to be copied from
GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2
An optional input parameter that specifies the version of the macro. PLISTVER
determines which parameter list the system generates. PLISTVER is an
optional input parameter on all forms of the macro, including the list form.
When using PLISTVER, specify it on all macro forms used for a request and
with the same value on all of the macro forms. The values are:
• IMPLIED_VERSION, which is the lowest version that allows all parameters
  specified on the request to be processed. If you omit the PLISTVER
  parameter, IMPLIED_VERSION is the default.
• MAX, if you want the parameter list to be the largest size currently possible.
  This size might grow from release to release and affect the amount of
  storage that your program needs.
  If you can tolerate the size change, IBM recommends that you always
  specify PLISTVER=MAX on the list form of the macro. Specifying MAX
  ensures that the list-form parameter list is always long enough to hold all
  the parameters you might specify on the execute form, when both are
  assembled with the same level of the system. In this way, MAX ensures that
  the parameter list does not overwrite nearby storage.
• 0, supports all parameters except those specifically referenced in higher
  versions.
• 1, supports both the following parameters and parameters from version 0:
  – CONVERTSIZE64
  – CONVERTSTART
  – GUARDSIZE64
  – GETSHARED
• 2, supports both the following parameters and parameters from version 0
  and 1:
  – GETCOMMON
  – PAGEPROTECT
  – PAGEUNPROTECT

To code: Specify one of the following:
• IMPLIED_VERSION
• MAX
IARV64 macro

- A decimal value of 0, 1 or 2.

\[MF=S\]
\[MF=(L,list \ addr)\]
\[MF=(L,list \ addr,attr)\]
\[MF=(L,list \ addr,0D)\]
\[MF=(E,list \ addr)\]
\[MF=(E,list \ addr,COMPLETE)\]
\[MF=(M,list \ addr)\]
\[MF=(M,list \ addr,COMPLETE)\]
\[MF=(M,list \ addr,NOCHECK)\]

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the P_LISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

Use the modify and execute forms in the following order:
- Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,list_addr,NOCHECK), to execute the macro.

\[list \ addr\]

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

\[attr\]

An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code \[attr\], the system provides a value of 0D.

\[COMPLETE\]

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

\[NOCHECK\]

Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.
REQUEST=PAGEFIX option of IARV64

REQUEST=PAGEFIX allows you to fix physical pages within one or more nonshared memory objects. It makes virtual storage areas, above the bar, reside in central storage (also called real storage) and ineligible for page-out while the address space specified by the ALETVALUE is swapped into central storage.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Can be used only by callers running in supervisor state or with PSW key 0-7.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts. Enabled or disabled for 64-bit common memory objects allocated with TYPE=DREF, or TYPE=PAGEABLE and the storage is in the first reference state.</td>
</tr>
<tr>
<td>Locks:</td>
<td>A local lock may be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space and can reside both above and below the bar.</td>
</tr>
</tbody>
</table>

Programming requirements

None

Restrictions

Pages that are fixed must be unfixed before the task owning the memory object terminates. Otherwise the address space where the memory object resides is terminated.

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>
IARV64 macro

15 Return code

When control returns to the caller, the ARs contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
</tr>
<tr>
<td>2-13</td>
</tr>
<tr>
<td>14-15</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=PAGEFIX option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=PAGEFIX</td>
<td></td>
</tr>
<tr>
<td>,LONG=YES</td>
<td>Default: LONG=YES</td>
</tr>
<tr>
<td>,LONG=NO</td>
<td></td>
</tr>
<tr>
<td>,RANGLIST=ranglist</td>
<td>ranglist: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=aletvalue</td>
<td>aletvalue: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=0</td>
<td>Default: ALETVALUE=0</td>
</tr>
<tr>
<td>,NUMRANGE=numrange</td>
<td>numrange: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,NUMRANGE=1</td>
<td>Default: NUMRANGE=1</td>
</tr>
<tr>
<td>,COND=NO</td>
<td>Default: COND=NO</td>
</tr>
<tr>
<td>,COND=YES</td>
<td></td>
</tr>
</tbody>
</table>
Parameters
A required parameter. REQUEST=PAGEFIX specifies that the data within the specified ranges be pagefixed.

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=PAGEFIX
A required parameter. REQUEST=PAGEFIX specifies that the data within the specified ranges be pagefixed.

PAGEFIX can only be requested for 64-bit private memory objects that are created using GETSTOR CONTROL=AUTH, and 64-bit common storage memory objects. PAGEFIX cannot be requested for 64-bit shared memory objects. 64-bit private or common storage memory objects backed by fixed 1 MB-page frames, or 64-bit private memory objects backed by fixed 2GB-page frames will be ignored.

PAGEFIX cannot be requested for a private memory object that was created using GETSTOR CONTROL=UNAUTH.

PAGEFIX cannot be requested for guard areas.

PAGEFIX specifies that the virtual storage areas are to reside in real storage and are ineligible for page-out while the address space is swapped in. This parameter does not prevent pages from being paged out when the entire address space is swapped out of real storage.

PAGEFIXed pages may be backed anywhere in real storage.

A page is considered PAGEFIXed until the number of valid PAGEUNFIXes issued for the page is equal to the number of valid PAGEFIXes previously issued for that page.
While a page is PAGEFIXed, the memory object, allocated with DETACHFIXED=NO, cannot be freed; if the system finds a PAGEFIXed area in the memory object, it abnormally ends the DETACH caller.

While a page is PAGEFIXed, the memory object allocated with DETACHFIXED=YES, can be freed successfully.

I/O can be done only to pages of memory objects that have been PAGEFIXed.

All I/O into virtual storage above the bar for an address space must be associated with the address space, that is, the ASID in the IOSB must be the ASID for the address space which owns the memory object. This is required so that I/O for the address space will be automatically purged during MEMTERM processing of the address space that owns the virtual storage above the bar or during I/O quiesce processing in preparation for swapping out the address space. The I/O must also be associated with the task which owns the memory object or one of its siblings. This is required so that all I/O is terminated and cleanup performed before the memory object is detached during task termination.

A resource manager must be provided to handle outstanding I/O when the task owning the memory object terminates. The resource manager must run before RSM’s task termination resource manager and must ensure that all I/O into the virtual storage above the bar is complete and any fixed storage is unfixed. This is required for both normal and abnormal task termination. For example, if the resource manager will be invoked through ABEND of the task termination if any virtual storage above the bar owned by the task is PAGEFIXED. This resource manager must ensure that all I/O into the memory object is complete. This is required for both normal and abnormal task termination.

PAGEFIX can be used only by callers running in supervisor state or with PSW key 0-7.

\texttt{\textbackslash{\texttt{LONG=\text{\texttt{YES}}}}}
\texttt{\textbackslash{\texttt{LONG=\text{\texttt{NO}}}}}

An optional input parameter that specifies whether the expected duration of the PAGEFIX is short or long. In general, a PAGEFIX is considered to be long if the time can be measured in seconds. The default is LONG=YES.

\texttt{\textbackslash{\texttt{LONG=\text{\texttt{YES}}}}}

The PAGEFIX is expected to be of a long duration.

\texttt{\textbackslash{\texttt{LONG=\text{\texttt{NO}}}}}

The PAGEFIX is expected to be of a short duration.

\texttt{\textbackslash{\texttt{RANGLIST=ranglist}}}

A required input parameter. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

\texttt{VSA}

Specifies the starting address of the data to be acted on.

The address specified must be within a memory object returned by GETSTOR CONTROL=AUTH or GETCOMMON. The value must always be on a physical 4 KB-page boundary.

The length of this field is 8 bytes.
IARV64 macro

Note: If the starting address is backed by pageable 1 MB-page frames, specify a value on a physical 1 MB-page boundary. Otherwise, demotion of pageable 1 MB-page frames to pageable 4 KB-page frames could occur.

NUMPAGES
Contains the number of physical 4 KB-pages in the area.

The number of 4 KB-pages specified starting with the specified VSA must lie within a single memory object.

The length of this field is 8 bytes.

Note: If the range includes addresses backed by pageable 1 MB-page frames, specify a number of 4 KB-pages that is a multiple of 256. Otherwise, demotion of pageable 1 MB-page frames to pageable 4 KB-page frames could occur.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,ALETVALUE=aletvalue

,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space in which the storage is to be pagefixed. The ALETVALUE parameter is ignored for 64-bit common memory objects.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,NUMRANGE=numrange

,NUMRANGE=1
An optional input parameter that specifies the number of entries in the supplied range list. The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,COND=NO

,COND=YES
An optional input parameter that specifies whether the request is unconditional or conditional. If you code COND=YES and there is insufficient storage to satisfy the request, instead of the request being abnormally ended the request will complete but a return code will be set to indicate that the request could not be completed successfully. In all cases the request will be abnormally ended for invalid requests, including violation of environmental restrictions. The default is COND=NO.

,COND=NO
The request is unconditional. The request is abnormally ended when the request cannot be satisfied.

,COND=YES
The request is conditional. The request is not abnormally ended for resource unavailability.
IARV64 macro

,RETCODE=retcode
An optional output parameter into which the return code is to be copied from
GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=rsncode
An optional output parameter into which the reason code is to be copied from
GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4, 5
An optional input parameter that specifies the version of the macro. PLISTVER
determines which parameter list the system generates. PLISTVER is an
optional input parameter on all forms of the macro, including the list form.
When using PLISTVER, specify it on all macro forms used for a request and
with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters
  specified on the request to be processed. If you omit the PLISTVER
  parameter, IMPLIED_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible.
  This size might grow from release to release and affect the amount of
  storage that your program needs.
  If you can tolerate the size change, IBM recommends that you always
  specify PLISTVER=MAX on the list form of the macro. Specifying MAX
  ensures that the list-form parameter list is always long enough to hold all
  the parameters you might specify on the execute form, when both are
  assembled with the same level of the system. In this way, MAX ensures that
  the parameter list does not overwrite nearby storage.

- **0**, supports all parameters except those specifically referenced in higher
  versions.

- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARE

- **2**, supports both the following parameters and parameters from version 0
  and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERSID
IARV64 macro

- OWNERCOM
- TYPE
- UNLOCKED
- USERTOKEN
- V64COMMON

- 3, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO

- 4, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

- 5, supports both the following parameters and parameters from versions 0, 1, 2, 3, 4:
  - UNITS
  - UNITSIZE

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4, or 5

,MF=S
,MF=(L,list_addr)
,MF=(L,list_addr,attr)
,MF=(L,list_addr,OD)
,MF=(E,list_addr)
,MF=(E,list_addr,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list_addr
   The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
   An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter
IARV64 macro

list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

\texttt{\textbackslash COMPLETE}

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

**REQUEST=PAGEUNFIX option of IARV64**

Use REQUEST=PAGEUNFIX to unfix physical pages within one or more nonshared or common memory objects.

**Environment**

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Can be used only by callers running in supervisor state or with PSW key 0-7.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>A local lock may be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space and can reside both below and above the bar.</td>
</tr>
</tbody>
</table>

**Programming requirements**

None

**Restrictions**

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

**Input register information**

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

**Output register information**

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:
Register

Contents

0-1   Used as work registers by the system

2-13  Unchanged

14-15 Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=PAGEUNFIX option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=PAGEUNFIX</td>
<td></td>
</tr>
<tr>
<td>,RANGLIST=ranglist</td>
<td>ranglist: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=aletvalue</td>
<td>aletvalue: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=0</td>
<td>Default: ALETVALUE=0</td>
</tr>
<tr>
<td>,NUMRANGE=numrange</td>
<td>numrange: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,NUMRANGE=1</td>
<td>Default: NUMRANGE=1</td>
</tr>
<tr>
<td>,COND=NO</td>
<td>Default: COND=NO</td>
</tr>
<tr>
<td>,COND=YES</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
IARV64 macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>.PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>.PLISTVER=0, 1, 2, 3, 4, 5</td>
<td></td>
</tr>
<tr>
<td>.MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>.MF=(L,list addr)</td>
<td><em>list addr</em>: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td>.MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>.MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>.MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>.MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

*name*

An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

*REQUEST=PAGEUNFIX*

A required parameter. REQUEST=PAGEUNFIX specifies that a range of storage has no I/O in progress and will no longer be used for I/O or will no longer be referenced disabled.

PAGEUNFIX can only be requested for 64-bit private memory objects that are created using GETSTOR CONTROL=AUTH, and 64-bit common storage memory objects. PAGEUNFIX cannot be requested for 64-bit shared memory objects. 64-bit private or common storage memory objects backed by fixed 1 MB-page frames, or 64-bit private memory objects backed by fixed 2GB-page frames will be ignored.

A page is considered PAGEFIXed until the number of valid PAGEUNFIXes issued for the page is equal to the number of valid PAGEFIXes previously issued for that page.

If a PAGEUNFIX is issued for a page that is not PAGEFIXed, the caller will be abnormally ended.

The PAGEUNFIX keyword can be used only by callers running in supervisor state or with PSW key 0-7.

*RANGLIST=ranglist*

A required input parameter, of a range list. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

*VSA*

denotes the starting address of the data to be acted on.

The address specified must be within a memory object returned by GETSTOR CONTROL=AUTH.

The value must always be on a physical 4 KB-page boundary.
The length of this field is 8 bytes.

**Note:** If the starting address is backed by pageable 1 MB-page frames, the number of 4 KB-pages specified must always be on a physical 1M-page boundary.

**NUMPAGES**
Contains the number of physical 4 KB-pages in the area.

The number of 4 KB-pages specified starting with the specified VSA must lie within a single memory object.

The length of this field is 8 bytes.

**Note:** If the range includes addresses backed by pageable 1 MB-page frames, the number of 4 KB-pages specified must be a multiple of 256.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,ALETVALUE=aletvalue
,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space in which the storage is to be unfixed.

The only supported values are 0 (primary) and 2 (home). ALETVALUE can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,NUMRANGE=numrange
,NUMRANGE=1
An optional input parameter that specifies the number of entries in the range list. The value specified must be no greater than 16. The default is 1.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,COND=NO
,COND=YES
This is an optional input parameter that specifies whether the request is unconditional or conditional. If you code COND=YES and there are unfixed pages in the range specified, instead of the request being abnormally ended, the request will complete but a return code will be set to indicate that the request was completed abnormally. In this case, the unfixed pages skipped and all the fixed pages will be unfixed. In all cases, the request will be abnormally ended for invalid requests including violations of environmental restrictions. The DEFAULT value is NO.

COND=NO
The request is unconditional. The request is abnormally ended when the request cannot be satisfied.

COND=YES
The request is conditional. The request is not abnormally ended for unfixed pages in the range specified.

,RETCODE=retcode
An optional output parameter into which the return code is to be copied from GPR 15.
IARV64 macro

**To code:** Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=rsncode

An optional output parameter into which the reason code is to be copied from GPR 0.

**To code:** Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION

,PLISTVER=MAX

,PLISTVER=0, 1, 2, 3, 4, 5

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, supports all parameters except those specifically referenced in higher versions.

- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED

- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
IARV64 macro

- USERTOKEN
- V64COMMON
- 3, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- 4, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE
- 5, supports both the following parameters and parameters from versions 0, 1, 2, 3, 4:
  - UNITS
  - UNITSIZE

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4, or 5

,MF=S
,MF=(L,list addr)
,MF=(L,list addr,attr)
,MF=(L,list addr,0D)
,MF=(E,list addr)
,MF=(E,list addr,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
   The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
   An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

,COMPLETE
   Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.
REQUEST=PAGEOUT option of IARV64

REQUEST=PAGEOUT notifies the system that data within physical pages of one or more memory objects will not be used in the near future.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and PSW key 8-15.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>A local lock may be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space and can reside both below and above the bar.</td>
</tr>
</tbody>
</table>

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None

**Syntax**

The REQUEST=PAGEOUT option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=PAGEOUT</td>
<td></td>
</tr>
<tr>
<td>,RANGLIST=ranglist</td>
<td><em>ranglist</em>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=aletvalue</td>
<td><em>aletvalue</em>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=0</td>
<td>Default: ALETVALUE=0</td>
</tr>
<tr>
<td>,NUMRANGE=numrange</td>
<td><em>numrange</em>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,NUMRANGE=1</td>
<td>Default: NUMRANGE=1</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td><em>retcode</em>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td><em>rsncode</em>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=0, 1, 3, 4</td>
<td></td>
</tr>
<tr>
<td>,MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td><em>list addr</em>: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td>,MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td></td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=PAGEOUT
A required parameter. REQUEST=PAGEOUT notifies the system that data within the specified ranges will not be used in the near future, i.e. for time measured in seconds (or longer), and are good candidates for paging.

Areas of the memory object that are PAGEFIXed or are in guard areas will not be affected.

RANGLIST=ranglist
A required input parameter. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

VSA
denotes the starting address of the data to be acted on.

The address specified must be within a memory object created by GETSTOR, GETSHARED, or GETCOMMON.

The value must always be on a physical 4 KB-page boundary.

The length of this field is 8 bytes.

Note: If the starting address is backed by pageable 1 MB-page frames, use a value that is on a physical segment boundary. Otherwise, demotion of pageable 1 MB-page frames to pageable 4 KB-page frames could occur.

NUMPAGES
Contains the number of physical 4 KB-pages in the area.

The number of 4 KB-pages specified starting with the specified VSA must lie within a single memory object.

The length of this field is 8 bytes.

Note: If the range includes addresses backed by pageable 1 MB-page frames, specify a number of 4 KB-pages in multiples of 256. Failure to do so might result in the demotion of pageable 1 MB-page frames to pageable 4 KB-page frames.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

ALETVALUE=aletvalue

An optional input parameter that indicates the ALET of the address space in which the virtual storage is to be paged out.
The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

\texttt{NUMRANGE=numrange}

An optional input parameter that specifies the number of entries in the supplied range list.

The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

\texttt{RETCODE=retcode}

An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

\texttt{RSNCODE=rsncode}

An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

\texttt{PLISTVER=IMPLIED\_VERSION}

\texttt{PLISTVER=MAX}

\texttt{PLISTVER=0, 1, 2, 3, 4}

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED\_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED\_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, supports all parameters except those specifically referenced in higher versions.

- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED

- **2**, supports both the following parameters and parameters from version 0 and 1:
IARV64 macro

- AMOUNTSIZE
- DETACHFIXED
- DOAUTHCHECKS
- DUMP
- DUMPPRIORITY
- DUMPPROTOCOL
- LOCALSYSAREA
- MEMLIMIT
- OPTIONVALUE
- ORDER
- OWNERASID
- OWNERCOM
- TYPE
- UNLOCKED
- USERTOKEN
- V64COMMON

- 3, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- 4, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4

\[\text{MF=S}\]
\[\text{MF=(L,list addr)}\]
\[\text{MF=(L,list addr,attr)}\]
\[\text{MF=(L,list addr,0D)}\]
\[\text{MF=(E,list addr)}\]
\[\text{MF=(E,list addr,COMPLETE)}\]

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant
IARV64 macro

code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

\texttt{list \textit{addr}}

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

\texttt{attr}

An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code \texttt{attr}, the system provides a value of 0D.

\texttt{COMPLETE}

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{REQUEST=PAGEIN option of IARV64} & \\
\hline
\texttt{REQUEST=PAGEIN} notifies the system that data within physical pages of one or more memory objects will be needed in the near future. & \\
\hline
\hline
\textbf{Environment} & \\
\hline
The requirements for the caller are: & \\
\hline
\begin{tabular}{|l|l|}
\textbf{Environmental factor} & \textbf{Requirement} \\
Minimum authorization: & Problem state and PSW key 8-15. \\
Dispatchable unit mode: & Task or SRB \\
Cross memory mode: & Any PASN, any HASN, any SASN \\
AMODE: & 31- or 64-bit \\
ASC mode: & Primary or access register (AR) \\
Interrupt status: & Enabled for I/O and external interrupts \\
Locks: & A local lock may be held. \\
Control parameters: & Control parameters must be in the primary address space and can reside both below and above the bar. \\
\end{tabular} & \\
\hline
\hline
\textbf{Programming requirements} & None \\
\hline
\textbf{Restrictions} & This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description. \\
\hline
\textbf{Input register information} & Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register. \\
\hline
\textbf{Output register information} & When control returns to the caller, the GPRs contain: \\
\begin{tabular}{|l|l|}
Register & Contents \\
\end{tabular} & \\
\hline
\end{tabular}
\end{table}
IARV64 macro

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

**Register**

**Contents**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None

**Syntax**

The REQUEST=PAGEIN option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=PAGEIN</td>
<td></td>
</tr>
<tr>
<td>,RANGLIST=ranglist</td>
<td>ranglist: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=aletvalue</td>
<td>aletvalue: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=0</td>
<td>Default: ALETVALUE=0</td>
</tr>
<tr>
<td>,NUMRANGE=numrange</td>
<td>numrange: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,NUMRANGE=1</td>
<td>Default: NUMRANGE=1</td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=PAGEIN
A required parameter. REQUEST=PAGEIN notifies the system that data within the specified ranges is needed in the near future and should be retrieved from auxiliary storage, if possible. An attempt to PAGEIN a range which contains a guard area will cause an ABEND.

RANGLIST=ranglist
A required input parameter. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

VSA
denotes the starting virtual address of the data to be acted on.

The virtual address specified must be within an allocated memory object returned by GETSTOR, GETSHARE, or GETCOMMON.

The value must always be on a physical 4 KB-page boundary.

The length of this field is 8 bytes.

NUPAGES
Contains the number of physical 4 KB-pages in the area.

The number of 4 KB-pages specified starting with the specified VSA must lie within a single memory object.

The length of this field is 8 bytes.
IARV64 macro

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,ALETVALUE=aletvalue
,
ALETVALUE=0
An optional input parameter that indicates the ALET of the space in which the virtual storage is to be paged in.
The only supported values are 0 (primary address space) and 2 (home address space). The ALETVALUE parameter may be used only by callers executing in supervisor state or with a system (0-7) PSW key. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,NUMRANGE=numrange
,NUMRANGE=1
An optional input parameter that specifies the number of entries in the supplied range list.
The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,RETCODE=retcode
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=rsncode
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- IMPLIED_VERSION, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- MAX, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- 0, supports all parameters except those specifically referenced in higher versions.
- 1, supports both the following parameters and parameters from version 0:
To code: Specify one of the following:
• IMPLIED_VERSION
• MAX
• A decimal value of 0, 1, 2, 3, 4

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The
**IARV64 macro**

List form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

```
,list addr
  The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
  An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

,COMPLETE
  Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.
```

**REQUEST=DISCARDDATA option of IARV64**

REQUEST=DISCARDDATA allows you to discard data within physical pages of one or more memory objects.

**Environment**

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state and PSW key 8-15.</td>
</tr>
<tr>
<td></td>
<td>The caller must be running in supervisor state or with PSW key 0-7 or have a PSW key that matches the storage key of the memory object to be cleared by DISCARDDATA.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Intercept status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>When a local lock is held for a request (GETSTOR, SHAREMEMOBJ, DETACH, or DISCARDDATA) the lock must be for the address space specified (or defaulted) by the input ALETVALUE.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be in the primary address space and can reside both below and above the bar.</td>
</tr>
</tbody>
</table>

**Programming requirements**

None

**Restrictions**

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.
**Input register information**

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

**Output register information**

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None

**Syntax**

The REQUEST=DISCARDDATA option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=DISCARDDATA</td>
<td></td>
</tr>
</tbody>
</table>
### IARV64 macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,KEEPREAL=YES</td>
<td>Default: KEEPREAL=YES</td>
</tr>
<tr>
<td>,KEEPREAL=NO</td>
<td></td>
</tr>
<tr>
<td>,CLEAR=YES</td>
<td>Default: CLEAR=YES</td>
</tr>
<tr>
<td>,CLEAR=NO</td>
<td></td>
</tr>
<tr>
<td>,RANGLIST=ranglist</td>
<td>ranglist: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=alevalue</td>
<td>alevalue: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=0</td>
<td>Default: ALETVALUE=0</td>
</tr>
<tr>
<td>,NUMRANGE=numrange</td>
<td>numrange: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,NUMRANGE=1</td>
<td>Default: NUMRANGE=1</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=0, 1, 2, 3, 4</td>
<td></td>
</tr>
<tr>
<td>,MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>,MF=(L,list_addr)</td>
<td>list_addr: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td>,MF=(L,list_addr,attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list_addr,0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list_addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list_addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

- **name**
  - An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

- **REQUEST=DISCARD_DATA**
  - A required parameter. REQUEST=DISCARD_DATA discards the data within the specified ranges.
For shared memory objects, the address space specified by the ALET or the default must have access to the memory object before issuing the DISCARDDATA (must have issued the IARV64 SHAREMEMOBJ prior to issuing the request).

Areas of the memory object that are PAGEFIXed, or are guard areas in the address space identified by the input ALET will not be discarded. If the DISCARDDATA service finds a PAGEFIXed, hidden, read-only, or guard area in the area to be discarded, the caller will be abnormally ended. However, any prior pages processed will have data in an indeterminate state when CLEAR=NO is used, and KEEPREAL=YES is also used or set as the default.

The caller must be in supervisor state or have PSW key 0-7 or have a PSW key that matches the storage key of the memory object to be cleared.

\textbf{KEEPREAL=\textit{YES}}

An optional parameter that specifies whether the real frames backing the pages to be discarded are to be freed or not. The default is KEEPREAL=YES.

\textbf{KEEPREAL=\textit{NO}}

The real frames backing the pages to be discarded are not to be freed unless there is shortage in real storage.

\textbf{CLEAR=\textit{YES}}

An optional parameter that specifies whether the data in the range should become binary zeros. The default is CLEAR=YES.

\textbf{CLEAR=\textit{NO}}

The data will be indeterminate.

\textbf{RANGLIST=\textit{ranglist}}

A required input parameter, of a range list. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

\textbf{VSA}

Denotes the starting address of the data to be acted on.

The address specified must be within a memory object returned by GETSTOR, GETSHARED, or GETCOMMON.

The value must always be on a physical 4 KB-page boundary.

The length of this field is 8 bytes.

\textbf{NUMPAGES}

Contains the number of physical 4 KB-pages in the area.

The number of 4 KB-pages specified starting with the specified VSA must lie within a single memory object.
IARV64 macro

The length of this field is 8 bytes.

Note: If the range includes addresses backed by pageable 1 MB-page frames, specify a number of 4 KB-pages that is a multiple of 256. Otherwise, demotion of pageable 1 MB-page frames to pageable 4 KB-page frames could occur.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,ALETVALUE=aletvalue
,ALETVALUE=0

An optional input parameter that indicates the ALET of the address space owning or with access to the memory object in which the virtual storage data is to be discarded.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,NUMRANGE=numrange
,NUMRANGE=1

An optional input parameter that specifies the number of entries in the supplied range list.

The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,RETCODE=retcode

An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSCODE=rsncode

An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- IMPLIED_VERSION, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- MAX, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX
ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- 0, supports all parameters except those specifically referenced in higher versions.
- 1, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- 2, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- 3, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- 4, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4
IARV64 macro

\*MF=(E,list addr,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

\*list addr

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

\*attr

An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

\*COMPLETE

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=CHANGEGUARD option of IARV64

IARV64 REQUEST=CHANGEGUARD requests that a specified amount of a private memory object be changed from the guard area to the usable area or vice versa. To avoid an abend for exceeding the MEMLIMIT, specify the COND=YES parameter.

IARV64 REQUEST=CHANGEGUARD only applies to 64-bit non-2GB private memory objects. If a 64-bit common memory object or a 64-bit shared memory object is specified on the request, a DC2 abend with reason code X'xx0058xx' is issued.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state and PSW key 8-15.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td></td>
<td>Note: The problem state caller running in PSW key 8-15 can use CHANGEGUARD only when the primary address space is the home address space.</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
</tbody>
</table>
IARV64 macro

Environmental factor

Locks:
A local lock may be held, subject to the following limitation:

When a local lock is held for a CHANGEGUARD request, the lock must be for the address space specified (or is set as the default) by the input ALETVALUE.

Control parameters:
Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements
None

Restrictions
This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information
Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None
### Syntax

The REQUEST=CHANGEGUARD option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td>IARV64</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=CHANGEGUARD</td>
<td>REQUEST=CHANGEGUARD</td>
</tr>
<tr>
<td><code>,CONVERT=TOGUARD</code></td>
<td><code>,CONVERT=TOGUARD</code></td>
</tr>
<tr>
<td><code>,CONVERT=FROMGUARD</code></td>
<td><code>,CONVERT=FROMGUARD</code></td>
</tr>
<tr>
<td><code>,MEMOBJSTART=memobjstart</code></td>
<td><code>memobjstart</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>,CONVERTSTART=convertstart</code></td>
<td><code>convertstart</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>,CONVERTSIZE=convertsize</code></td>
<td><code>convertsize</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>,CONVERTSIZE64=convertsize64</code></td>
<td><code>convertsize64</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>,RETCODE=retcode</code></td>
<td><code>retcode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,RSNCODE=rsncode</code></td>
<td><code>rsncode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>_PLISTVER=IMPLIED_VERSION_</code></td>
<td>Default: <code>PLISTVER=IMPLIED_VERSION</code></td>
</tr>
<tr>
<td><code>_PLISTVER=MAX</code></td>
<td></td>
</tr>
<tr>
<td><code>_PLISTVER=0, 1, 2, 3, 4</code></td>
<td></td>
</tr>
<tr>
<td><code>_MF=S</code></td>
<td>Default: <code>MF=S</code></td>
</tr>
<tr>
<td><code>_MF=(L, list addr)</code></td>
<td><code>list addr</code>: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td><code>_MF=(L, list addr,attr)</code></td>
<td></td>
</tr>
<tr>
<td><code>_MF=(L, list addr,0D)</code></td>
<td></td>
</tr>
<tr>
<td><code>_MF=(E, list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>_MF=(E, list addr,COMPLETE)</code></td>
<td></td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

name

An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=CHANGEGUARD

A required parameter. REQUEST=CHANGEGUARD changes the amount of guard area in the specified memory object. It changes part of the memory object from a guard area to a usable area, or vice versa.

If the CHANGEGUARD service finds a PAGEFIXed area in the area to be converted into a guard area, the caller will be abnormally ended. If a request is made to guard a guard area or to unguard an area that is not guarded a return code 04 will be issued.

If you code COND=YES and there is insufficient storage to satisfy the request, instead of the request being abnormally ended, the request will complete, but a return code will be set to indicate that the request could not be completed successfully.

For a problem state program running in PSW key (8–15), the PSW key of the caller must match the storage key of the memory object and the memory object must be owned by one of the following:

- The calling task
- The job step task
- An ancestor task up through the job step task

,CONVERT=TOGUARD

,CONVERT=FROMGUARD

A required parameter that specifies whether to add or remove guard areas.

,CONVERT=TOGUARD

Convert the specified amount of usable areas to the guard areas. The data in the converted areas will be released. This operation reduces the amount of virtual storage that contributes toward the MEMLIMIT for the address space identified by ALETVALUE. If CONVERTSTART is used then a guard area is created from a usable area starting with the address specified continuing for the number of segments specified by CONVERTSIZE. If CONVERTSTART is not used when GUARDLOC=LOW was specified on the GETSTOR request, the first usable virtual address space in the memory object is increased. If CONVERTSTART is not used when GUARDLOC=HIGH was specified on the GETSTOR request, the last usable virtual address space in the memory object is decreased.

,CONVERT=FROMGUARD

Convert the specified amount of guard area to be usable area. Any previously guarded pages that were converted as part of this request will appear as pages of zeros. Any pages that were already within a usable area will be unchanged. This operation increases the amount of area that contributes toward the MEMLIMIT for the address space designated by ALETVALUE.

If CONVERTSTART is used then a usable area is created from a guard area starting with the address specified continuing for the number of segments specified by CONVERTSIZE. If CONVERTSTART is not used when GUARDLOC=LOW is specified, the first usable virtual address space in
the memory object is decreased. If CONVERTSTART is not used when GUARDLOC=HIGH is specified, the last usable virtual address space in the memory object is increased.

**MEMOBJSTART=memobjstart**
MEMOBJSTART and CONVERTSTART are a set of mutually exclusive keys. This set is required; only one keyword must be specified. An input parameter that belongs to a required set of mutually exclusive keys. It is the name (RS-type), or address in register (2)-(12), of an eight-byte input that contains the address of the first byte in the memory object.

**CONVERTSTART=convertstart**
CONVERTSTART and MEMOBJSTART are a set of mutually exclusive keys. This set is required; only one keyword must be specified. An input parameter that belongs to a required set of mutually exclusive keys. CONVERTSTART specifies the address to add a guard area (continuing to the virtual address specified by adding the bytes defined in CONVERTSIZE to CONVERTSTART minus one) when CONVERT(TOGUARD) is requested, and specifies the address to remove from the guard area (continuing to the virtual address space specified by adding the bytes defined by CONVERTSIZE to CONVERTSTART minus one) when CONVERT(FROMGUARD) is requested.

Two contiguous guard areas will be consolidated into one contiguous guard area whenever possible. For example, if the guard area that was defined when the memory object was created is contiguous with a guard area created using CONVERTSTART, then the two guard areas are combined into one.

Specifying MEMOBJSTART will change the guard area only at the beginning or the end of the memory object. Whether the guard area is at the beginning or the end is specified on the IARV64 REQUEST=GETSTOR GUARDLOC=[HIGH|LOW]

IBM recommends that if CONVERTSTART is used to manage the guard areas within a memory object that all REQUEST=CHANGEGUARD use CONVERTSTART.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

**CONVERTSIZE=convertsize**
CONVERTSIZE and CONVERTSIZE64 are a set of mutually exclusive keys. This set is required; only one key must be specified. A fullword integer input parameter, that indicates the number of contiguous megabytes that should be removed from the guard area (FROMGUARD) or that should be changed to being part of the guard area (TOGUARD).

For CONVERT=TOGUARD and MEMOBJSTART, CONVERTSIZE or CONVERTSIZE64 must not be larger than the number of usable pages in the memory object to allow successful completion. For CONVERT=FROMGUARD, CONVERTSIZE must not be larger than the number of remaining pages in the default guard area of the memory object to allow successful completion.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a fullword field.

**CONVERTSIZE64=convertsize64**
CONVERTSIZE64 and CONVERTSIZE are a set of mutually exclusive keys. This set is required; only one key must be specified. A doubleword integer input parameter, that indicates the number of contiguous megabytes that should be removed from the guard area (FROMGUARD) or that should be changed to being part of the guard area (TOGUARD).
For CONVERT=TOGUARD and MEMOBJSTART, CONVERTSIZE or CONVERTSIZE64 must not be larger than the number of usable pages in the memory object to allow successful completion. For CONVERT=FROMGUARD, CONVERTSIZE must not be larger than the number of remaining pages in the default guard area of the memory object to allow successful completion.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

\COND=NO
\COND=YES

An optional input parameter that specifies whether the request is unconditional or conditional. When you code COND=YES and there is insufficient storage to satisfy the request, instead of the request being abnormally ended the request will complete but a return code will be set to indicate that the request could not be completed successfully. In all cases the request will be abnormally ended for invalid requests, including violation of environmental restrictions. The default is COND=NO.

\COND=NO
The request is unconditional. The request will be abnormally ended when the request cannot be satisfied.

\COND=YES
The request is conditional. The request will not be abnormally ended when a MEMLIMIT violation occurs.

\ALETVALUE=aletvalue
\ALETVALUE=0
An optional input parameter that indicates the ALET of the address space in which the memory object resides.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

\RETCODE=retcode
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

\RSNCODE=rsncoed
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

\PLISTVER=IMPLIED_VERSION
\PLISTVER=MAX
\PLISTVER=0, 1, 2, 3, 4
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- IMPLIED_VERSION, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
IARV64 macro

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, supports all parameters except those specifically referenced in higher versions.

- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, supports all parameters except those specifically referenced in higher versions.

- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED

- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMP PRIORITY
  - DUMP PROTOCOL
  - LOCALSYSAREA
  - MEM LIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
IARV64 macro

- TYPE
- UNLOCKED
- USERTOKEN
- V64COMMON
- 3, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- 4, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4

\[MF=S\]
\[MF=\{L, list addr\}\]
\[MF=\{L, list addr, attr\}\]
\[MF=\{L, list addr, 0D\}\]
\[MF=\{E, list addr\}\]
\[MF=\{E, list addr, COMPLETE\}\]

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

\[list addr\]

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

\[attr\]

An optional 1- to 60 character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code \[attr\], the system provides a value of 0D.

\[COMPLETE\]

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.
REQUEST=PROTECT option of IARV64

REQUEST=PROTECT requests that data within one or more memory objects be made read-only.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and PSW key 8-15.</td>
</tr>
<tr>
<td></td>
<td>The caller must have a PSW key of 0 or a PSW key that matches the storage to be protected.</td>
</tr>
<tr>
<td></td>
<td>The caller must be running in supervisor state or with PSW key 0-7 to use the ALETVALUE keyword.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts for 64-bit private.</td>
</tr>
<tr>
<td></td>
<td>Enabled or disabled for I/O and external interrupts for 64-bit common storage.</td>
</tr>
<tr>
<td>Locks:</td>
<td>You may hold the local lock for the target address space. If you hold the local lock, you may also hold the CMS lock. For disabled callers, no spin locks higher than the RSM locks can be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space and can reside both below and above the bar.</td>
</tr>
</tbody>
</table>

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>
When control returns to the caller, the ARs contain:

**Register**

**Contents**

0-1 Used as work registers by the system

2-13 Unchanged

14-15 Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None

**Syntax**

The REQUEST=PROTECT option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=PROTECT</td>
<td></td>
</tr>
<tr>
<td>,RANGLIST=ranglist</td>
<td>ranglist: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=aletvalue</td>
<td>aletvalue: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=0</td>
<td>Default: ALETVALUE=0</td>
</tr>
<tr>
<td>,NUMRANGE=numrange</td>
<td>numrange: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,NUMRANGE=1</td>
<td>Default: NUMRANGE=1</td>
</tr>
<tr>
<td>,AMOUNTSIZE=1MEG</td>
<td>Default: AMOUNTSIZE=4K. AMOUNTSIZE=1MEG is the default when the data to be acted on is from a fixed 1 MB-page frames request. AMOUNTSIZE cannot be specified when the data to be acted on is from a fixed 2 GB-page frames request.</td>
</tr>
<tr>
<td>,AMOUNTSIZE=4K</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
IARV64 macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=0, 1, 2, 3, 4, 5</td>
<td></td>
</tr>
<tr>
<td>MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>MF=(L, list addr)</td>
<td>list addr: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td>MF=(L, list addr, attr)</td>
<td></td>
</tr>
<tr>
<td>MF=(L, list addr, 0D)</td>
<td></td>
</tr>
<tr>
<td>MF=(E, list addr)</td>
<td></td>
</tr>
<tr>
<td>MF=(E, list addr, COMPLETE)</td>
<td></td>
</tr>
<tr>
<td>MF=(M, list addr)</td>
<td></td>
</tr>
<tr>
<td>MF=(M, list addr, COMPLETE)</td>
<td></td>
</tr>
<tr>
<td>MF=(M, list addr, NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=PROTECT
A required parameter. REQUEST=PROTECT specifies that a range of virtual storage be made read-only.

Areas of the memory object that are in guard areas or hidden will not be affected.

,RANGLIST=ranglist
A required input parameter. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

VSA
Denotes the starting address of the data to be acted on.

The address specified must be within a created memory object returned by GETSTOR or GETCOMMON.

The value must be on a segment (1M) boundary when either:
  • PAGEFRAMESIZE=1MEG | 1M is specified, or
  • PAGEFRAMESIZE=MAX is specified and the address specified is backed by fixed 1M page frames.

The value must be on a region (2G) boundary when PAGEFRAMESIZE=2G is specified. Otherwise, the value must be on a page (4K) boundary.
Note: If the range includes addresses backed by pageable 1 MB-page frames, specify a value on a physical segment boundary. Otherwise, demotion of pageable 1 MB-page frames to pageable 4 KB-page frames could occur.

The length of this field is 8 bytes.

**AMOUNT**
Contains the number of 4K, 1M, or 2G pages to be acted on.

The number of 4K, 1M, or 2G pages specified starting with the specified VSA must lie within a single memory object.

The length of this field is 8 bytes.

The amount contains the number of 1M pages to be acted on when the specified VSA is backed by fixed large page frames (1M).

The amount contains the number of 2G pages to be acted on when the specified VSA is backed by fixed 2G page frames (2G).

Note: If the range includes addresses backed by pageable 1M page frames, specify a number of 4K pages in multiples of 256; otherwise, demotion of pageable 1M page frames to pageable 4K page frames could occur.

**AMOUNTSIZE=4K**
**AMOUNTSIZE=1MEG**

An optional input parameter that specifies how the AMOUNT value specified on the RANGLIST parameter is treated. The default value is AMOUNTSIZE=4K.

When the data to be acted on is from a fixed 1M page frames request (TYPE=FIXED and PAGEFRAMESIZE=1M is specified, or TYPE is not specified and PAGEFRAMESIZE=1MEG is specified, or PAGEFRAMESIZE=MAX is specified and the address specified is backed by fixed 1M page frames), the default is AMOUNTSIZE=1MEG. Only AMOUNTSIZE=1MEG can be specified.

When the data to be acted on is from a fixed 2G page frames request (TYPE=FIXED and PAGEFRAMESIZE=2G is specified), AMOUNTSIZE cannot be specified. The AMOUNT value specified on the RANGLIST parameter is treated as the number of 2G pages to be acted on.

**4K** The AMOUNT value specified on the RANGLIST parameter is the number of 4K page frames to be acted on.

**1MEG**

The AMOUNT value specified on the RANGLIST parameter is the number of 1M page frames to be acted on.

**ALETVALUE=aletvalue**
**ALETVALUE=0**

An optional input parameter that indicates the ALET of the address space in which the virtual storage is to be protected.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a fullword field.

**NUMRANGE=numrange**
**IARV64 macro**

**,NUMRANGE=I**

An optional input parameter that specifies the number of entries in the supplied range list.

The value specified must be no greater than 16.

Default: 1

**,RETCODE=retcode**

An optional output parameter into which the return code is to be copied from GPR 15.

**To code:** Specify the RS-type address of a fullword field, or register (2)-(12).

**,RSNCODE=rsncode**

An optional output parameter into which the reason code is to be copied from GPR 0.

**To code:** Specify the RS-type address of a fullword field, or register (2)-(12).

**,PLISTVER=IMPLIED_VERSION**

**,PLISTVER=MAX**

**,PLISTVER=0, 1, 2, 3, 4, 5**

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form.

When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM suggests that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, supports all parameters except those specifically referenced in higher versions.

- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - GETSHARE

- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
IARV64 macro

- LOCALSYSAREA
- MEMLIMIT
- OPTIONVALUE
- ORDER
- OWNERASID
- OWNERCOM
- TYPE
- UNLOCKED
- USERTOKEN
- V64COMMON

- 3, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO

- 4, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

- 5, supports both the following parameters and parameters from versions 0, 1, 2, 3, 4:
  - UNITS
  - UNITSIZE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4, or 5

, MF=S
, MF=(L, list addr)
, MF=(L, list addr, attr)
, MF=(L, list addr, 0D)
, MF=(E, list addr)
, MF=(E, list addr, COMPLETE)
, MF=(M, list addr)
, MF=(M, list addr, COMPLETE)
, MF=(M, list addr, NOCHECK)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant
IARV64 macro

code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:

- Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,list_addr,NOCHECK), to execute the macro.

\texttt{list addr}

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

\texttt{attr}

An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code \texttt{attr}, the system provides a value of 0D.

\texttt{COMPLETE}

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

\texttt{NOCHECK}

This parameter specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

REQUEST=UNPROTECT option of IARV64

REQUEST=UNPROTECT requests that data within one or more memory objects be made modifiable.

The IARV64 REQUEST=UNPROTECT unprotects pages/segments within 64-Bit Private or 64-Bit Common memory objects. 64-bit fixed 2 GB private memory objects cannot be protected.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and PSW key 8-15.</td>
</tr>
<tr>
<td></td>
<td>The caller must have a PSW key of 0 or a PSW key that matches the storage to be unprotected.</td>
</tr>
<tr>
<td></td>
<td>The caller must be running in supervisor state or with PSW key 0-7 to use the ALETVALUE keyword.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
</tbody>
</table>
Environmental factor  Requirement

ASC mode:  Primary or access register (AR)
Interrupt status:  Enabled for I/O and external interrupts for 64-bit private.

Enabled or disabled for I/O and external interrupts for 64-bit common storage.
Locks:

You may hold the local lock for the target address space. If you hold the local lock, you may also hold the CMS lock.
For disabled callers no spin locks higher than the RSM locks can be held.

Control parameters:  Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register  Contents

0  Reason code, if GPR 15 is non-zero
1  Used as a work register by the system
2-13  Unchanged
14  Used as a work register by the system
15  Return code

When control returns to the caller, the ARs contain:

Register  Contents

0-1  Used as work registers by the system
2-13  Unchanged
14-15  Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None
## Syntax

The REQUEST=UNPROTECT option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=UNPROTECT</td>
<td></td>
</tr>
<tr>
<td>,RANGLIST=ranglist</td>
<td>ranglist: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=alevalue</td>
<td>aletvalue: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,ALETVALUE=0</td>
<td><strong>Default:</strong> ALETVALUE=0</td>
</tr>
<tr>
<td>,NUMRANGE=numrange</td>
<td>numrange: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,NUMRANGE=1</td>
<td><strong>Default:</strong> NUMRANGE=1</td>
</tr>
<tr>
<td>,AMOUNTSIZE=1MEG</td>
<td><strong>Default:</strong> AMOUNTSIZE=4K. AMOUNTSIZE=1MEG is the default when the data to</td>
</tr>
<tr>
<td>,AMOUNTSIZE=4K</td>
<td>be acted on is from a fixed 1 MB-page frames request. AMOUNTSIZE cannot be</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td><strong>Default:</strong> PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=0, 1, 2, 3, 4, 5</td>
<td></td>
</tr>
<tr>
<td>,MF=S</td>
<td><strong>Default:</strong> MF=S</td>
</tr>
<tr>
<td>,MF=(L, list addr)</td>
<td>list addr: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td>,MF=(L, list addr, attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L, list addr, 0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E, list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E, list addr, COMPLETE)</td>
<td></td>
</tr>
<tr>
<td>,MF=(M, list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(M, list addr, COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>
Syntax Description

\[ \text{,MF=}(M, \text{list addr}, \text{NOCHECK}) \]

Parameters

The parameters are explained as follows:

- **name**
  A required parameter. REQUEST=UNPROTECT specifies that a range of virtual storage be made modifiable.

- **REQUEST=UNPROTECT**
  A required parameter. REQUEST=UNPROTECT specifies that a range of virtual storage be made modifiable.
  Areas of the memory object that are in guard areas or hidden will not be affected.

- **,RANGLIST=ranglist**
  A required input parameter. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

  - **VSA**
    Denotes the starting address of the data to be acted on.
    The address specified must be within a created memory object returned by GETSTOR or GETCOMMON.
    The value must be on a segment (1M) boundary when either:
    - PAGEFRAMESIZE=1MEG|1M is specified, or
    - PAGEFRAMESIZE=MAX is specified and the address specified is backed by fixed 1M page frames.
    The value must be on a region (2G) boundary when PAGEFRAMESIZE=2G is specified. Otherwise, the value must be on a page (4K) boundary.
    **Note:** If the range includes addresses backed by pageable 1 MB-page frames, specify a value on a physical segment boundary. Otherwise, demotion of pageable 1 MB-page frames to pageable 4 KB-page frames could occur.
    The length of this field is 8 bytes.

  - **AMOUNT**
    Contains the number of 4K, 1M, or 2G pages to be acted on.
    The number of 4K, 1M, or 2G pages specified starting with the specified VSA must lie within a single memory object.
    The length of this field is 8 bytes.
    The amount contains the number of 1M pages to be acted on when the specified VSA is backed by fixed large page frames (1M).
    The amount contains the number of 2G pages to be acted on when the specified VSA is backed by fixed 2G page frames (2G).
Note: If the range includes addresses backed by pageable 1M page frames, specify a number of 4K pages in multiples of 256; otherwise, demotion of pageable 1M page frames to pageable 4K page frames could occur.

\texttt{AMOUNTSIZE=4K}
\texttt{AMOUNTSIZE=1MEG}

An optional input parameter that specifies how the AMOUNT value specified on the RANGLIST parameter is treated. The default value is AMOUNTSIZE=4K.

When the data to be acted on is from a fixed 1M page frames request (TYPE=FIXED and PAGEFRAMESIZE=1M is specified, or TYPE is not specified and PAGEFRAMESIZE=1MEG is specified, or PAGEFRAMESIZE=MAX is specified and the address specified is backed by fixed 1M page frames), the default is AMOUNTSIZE=1MEG. Only AMOUNTSIZE=1MEG can be specified.

When the data to be acted on is from a fixed 2G page frames request (TYPE=FIXED and PAGEFRAMESIZE=2G is specified), AMOUNTSIZE cannot be specified. The AMOUNT value specified on the RANGLIST parameter is treated as the number of 2G pages to be acted on.

\textbf{4K} The AMOUNT value specified on the RANGLIST parameter is the number of 4K page frames to be acted on.

\textbf{1MEG} The AMOUNT value specified on the RANGLIST parameter is the number of 1M page frames to be acted on.

\texttt{ALETVALUE=aletvalue}
\texttt{ALETVALUE=0}

An optional input parameter that indicates the ALET of the address space in which the virtual storage is to be unprotected.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

\texttt{NUMRANGE=numrange}
\texttt{NUMRANGE=1}

An optional input parameter that specifies the number of entries in the supplied range list.

The value specified must be no greater than 16. The default is 1.

\texttt{RETCODE=retcode}

An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

\texttt{RSNCODE=rsncode}

An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

\texttt{PLISTVER=IMPLIED_VERSION}
\texttt{PLISTVER=MAX}
\texttt{PLISTVER=0, 1, 2, 3, 4, 5}

An optional input parameter that specifies the version of the macro. PLISTVER
IARV64 macro

determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, you can always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - GETCOMMON
  - PAGEPROTECT
  - PAGEUNPROTECT
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERSID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
IARV64 macro

- TRACKINFO
- 4, supports both the following parameter and parameters from versions 0, 1, 2, 3:
- DMAPAGETABLE
- 5, supports both the following parameters and parameters from versions 0, 1, 2, 3, 4:
- UNITS
- UNITSIZE

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4, or 5

\[MF=S\]
\[MF=(L, list addr)\]
\[MF=(L, list addr, attr)\]
\[MF=(L, list addr, 0D)\]
\[MF=(E, list addr)\]
\[MF=(E, list addr, COMPLETE)\]
\[MF=(E, list addr, NOCHECK)\]

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

You can use the modify and execute forms in the following order:
- Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,list_addr,NOCHECK), to execute the macro.

\[list addr\]

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

\[attr\]

An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter
IARV64 macro

list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr, the system provides a value of 0D.

**COMPLETE**

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

**NOCHECK**

This parameter specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

**REQUEST=LIST option of IARV64**

REQUEST=LIST requests a list of objects be provided to the caller.

**Environment**

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Minimum authorization:</strong></td>
<td>Problem state and PSW key 8-15.</td>
</tr>
</tbody>
</table>

The caller must be running in supervisor state or with PSW key 0-7 to use the following parameters:

- **GETSTOR**
  - KEY
  - CONTROL=AUTH
  - ALETVALUE
  - PAGEFRAMESIZE=1M/MAX
- **V64SHARE**
- **DETACH**
  - AFFINITY=SYSTEM
  - OWNER=NO
- **PAGEFIX**
- **PAGEUNFIX**
- **LIST**
- **SHAREMEMOBJ**
- **CHANGEACCESS**

The caller must be running in supervisor state or with PSW key 0-7 or have a PSW key that matches the storage key of the memory object to be cleared by DISCARDDATA.

The caller must be running in supervisor state or with PSW key 0-7 to DETACH a memory object owned by another task.

**Dispatchable unit mode:** Task or SRB.

**Cross memory mode:** Any PASN, any HASN, any SASN.

**Note:** The problem state caller running in PSW key 8-15 can use GETSTOR/DETACH only when the primary address space is the home address space.

**AMODE:** 31- or 64-bit.

**ASC mode:** Primary or access register (AR)

**Interrupt status:** Enabled for I/O and external interrupts

**Locks:** A local lock may be held, subject to the following limitation:

When a local lock is held for a requests (GETSTOR, SHAREMEMOBJ, DETACH, or DISCARDDATA) for non-shared memory objects, the lock must be for the address space specified (or defaulted) by the input ALETVALUE.
IARV64 macro

Environmental factor
Control parameters: Requirement
Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements
None

Restrictions
This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information
Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

Register | Contents
---|---
0 | Reason code, if GPR 15 is non-zero
1 | Used as a work register by the system
2-13 | Unchanged
14 | Used as a work register by the system
15 | Return code

When control returns to the caller, the ARs contain:

Register | Contents
---|---
0-1 | Used as work registers by the system
2-13 | Unchanged
14-15 | Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None

Syntax
The REQUEST=LIST option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>
## Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=LIST</td>
<td></td>
</tr>
<tr>
<td>,V64LISTPTR=v64listptr</td>
<td><em>v64listptr</em>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,V64LISTLENGTH=v64listlength</td>
<td><em>v64listlength</em>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,V64SHARED=NO</td>
<td>Default: V64SHARED=NO</td>
</tr>
<tr>
<td>,V64SHARED=YES</td>
<td></td>
</tr>
<tr>
<td>,V64COMMON=NO</td>
<td>Default: V64COMMON=NO</td>
</tr>
<tr>
<td>,V64COMMON=YES</td>
<td></td>
</tr>
<tr>
<td>,TRACKINFO=NO</td>
<td>Default: TRACKINFO=NO</td>
</tr>
<tr>
<td>,TRACKINFO=YES</td>
<td></td>
</tr>
<tr>
<td>,V64SELECT=NO</td>
<td>Default: V64SELECT=NO</td>
</tr>
<tr>
<td>,V64SELECT=YES</td>
<td></td>
</tr>
<tr>
<td>,USERTOKEN=usertoken</td>
<td>Default: USERTOKEN=NO_USERTKN</td>
</tr>
<tr>
<td>,USERTOKEN=NO_USERTOKEN</td>
<td></td>
</tr>
<tr>
<td>,SVCDUMPRGN=YES</td>
<td>Default: SVCDUMPRGN=YES</td>
</tr>
<tr>
<td>,SVCDUMPRGN=NO</td>
<td></td>
</tr>
<tr>
<td>,SVCDUMPRGN=ALL</td>
<td></td>
</tr>
<tr>
<td>,DUMP=ALL</td>
<td>Default: DUMP=ALL</td>
</tr>
<tr>
<td>,DUMP=LIKECSA</td>
<td></td>
</tr>
<tr>
<td>,DUMP=LIKESQA</td>
<td></td>
</tr>
<tr>
<td>,DUMP=LIKERGN</td>
<td></td>
</tr>
<tr>
<td>,DUMPPROTOCOLED=NO</td>
<td>Default: DUMPPROTOCOLED=NO</td>
</tr>
<tr>
<td>,ORDER=ASCENDING</td>
<td>Default: ORDER=ASCENDING</td>
</tr>
<tr>
<td>,ORDER=DUMPPRIORITY</td>
<td></td>
</tr>
<tr>
<td>,DUMPPROTOCOLED=YES</td>
<td></td>
</tr>
</tbody>
</table>
### IARV64 Macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,OWNERCOM=ALL</td>
<td>Default: OWNERCOM=ALL</td>
</tr>
<tr>
<td>,OWNERCOM=HOME</td>
<td></td>
</tr>
<tr>
<td>,OWNERCOM=PRIMARY</td>
<td></td>
</tr>
<tr>
<td>,OWNERCOM=SYSTEM</td>
<td></td>
</tr>
<tr>
<td>,OWNERCOM=BYASID</td>
<td></td>
</tr>
<tr>
<td>,OWNERASID=ownerasid</td>
<td>,OWNERASID=ALL</td>
</tr>
<tr>
<td>,PAGEFRAMESIZE=All</td>
<td>Default: PAGEFRAMESIZE=All</td>
</tr>
<tr>
<td>,PAGEFRAMESIZE=4K</td>
<td></td>
</tr>
<tr>
<td>,PAGEFRAMESIZE=1MEG</td>
<td></td>
</tr>
<tr>
<td>,OWNERJOBNAME=ownerjobname</td>
<td>ownerjobname: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,OWNERJOBNAME=ALL</td>
<td>Default: OWNERJOBNAME=ALL</td>
</tr>
<tr>
<td>,ATTRIBUTE=DEFS</td>
<td>Default: ATTRIBUTE=DEFS</td>
</tr>
<tr>
<td>,ATTRIBUTE=NOTOWNERGONE</td>
<td></td>
</tr>
<tr>
<td>,ATTRIBUTE=OWNERGONE</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=0, 1, 2, 3, 4, 5</td>
<td></td>
</tr>
<tr>
<td>,MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>,MF=(L, list addr)</td>
<td>list addr: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td>,MF=(L, list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L, list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E, list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E, list addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=LIST
A required parameter. REQUEST=LIST provides information about memory objects.

The information is returned in a work area that you specify, in a format described in IAXV64WA. Information includes starting address, ending address, storage key and flags indicating if the memory object is shared or if it contains multiple guard areas.

The following information can be requested:
- Memory objects for the entire address space
- Memory objects in the entire address space that have been marked SVCDUMPRGN=YES
- Memory objects in the entire address space that have a specific SVCDUMPRGN attribute
- Memory objects in the entire address space that have a specific PAGEFRAMESIZE attribute
- Shared memory objects for the entire system
- 64-bit common memory object for the entire system.

,V64LISTPTR=v64listptr
A required input parameter that contains the address that specifies the address of the work area which contains the results of the list request. This work area must be in fixed storage addressable from the address space for which the LIST request is made, and must be initialized to zero by the caller.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,V64LISTLENGTH=v64listlength
A required input parameter that specifies the length of the work area which contains the results of the list request. The work area must be at least 108 bytes long when TRACKINFO=YES or 64 bytes long when TRACKINFO=NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,V64SHARED=NO
,V64SHARED=YES
An optional input parameter that specifies whether the list of memory objects returned is for private memory objects, which the current primary space owns as well as shared memory objects connected to the current primary address space, or a list of all shared memory objects defined in the system through GETSHARED. The default is V64SHARED=NO.

,V64SHARED=NO
The list of memory objects returned for the current primary address space includes private memory objects that are defined for the private area through an IARV64 GETSTOR and shared memory objects connected to the current primary address space through an IARV64 SHAREMEMOBJ.

,V64SHARED=YES
The list is of shared memory objects defined for the system through GETSHARED.
An optional parameter that specifies whether the list of memory object returned is for the current primary address space, or a list of all 64-bit common memory objects allocated in the system via an IARV64 REQUEST=GETCOMMON. The default is V64COMMON=NO.

The list of memory objects returned for the current primary address space includes private memory objects (which are defined for the private area via an IARV64 REQUEST=GETSTOR), shared memory objects (connected to the current primary address space via an IARV64 REQUEST=SHAREMEMOBJ).

The list of memory objects returned contains all 64-bit common memory objects defined in the system via IARV64 REQUEST=GETCOMMON.

An optional parameter that specifies whether the list request is for all allocated memory objects or for a subset of the allocated memory objects. The default is V64SELECT=NO.

The request is for all allocated memory objects. No additional selection criteria apply to the list of memory objects returned.

The request is for a subset of the allocated memory objects. Only memory objects that meet all the selection criteria is returned. If a selection criteria keyword is not specified, its default will apply. If no memory object meets the selection criteria, no object will be returned.

When V64COMMON=YES is specified, an optional parameter that specifies if the common memory object tracking information should be returned. Default is TRACKINFO=NO.

Common memory object tracking information will not be returned.

Common memory object tracking information will be returned. See IAXV64WA for output format.

When V64SELECT=YES is specified, an optional parameter that specifies whether additional selection criteria based on user token is applied to the set of memory object descriptions returned by the LIST request. The default is NO_USERTKN.

The memory objects returned are not filtered based on USERTKN. All memory objects, regardless of what the USERTKN specification was on the IARV64 GETSTOR or IARV64 SHAREMEMOBJ request for V64SHARED=NO or IARV64 GETSHARED request for V64SHARED=YES, are included in the set of memory objects returned.
When V64SHARED=NO is specified, memory objects in the current primary address space that have a matching user token specified on the IARV64 GETSTOR or IARV64 SHAREMEMOBJ request are included in the set of memory objects returned. When V64SHARED=YES is specified, shared memory objects defined in the system that have a matching user token specified on the IARV64 GETSHARED request are included in the set of memory objects returned.

When V64SELECT=YES is specified, an optional parameter that specifies whether the memory object should be included within the set of memory object descriptions returned by the LIST request. The default is SVCDUMPRGN=YES. This keyword is ignored when V64SHARED=YES is specified.

SVCDUMPRGN=YES
The memory objects with the SVCDUMPRGN=YES attribute are included in the set of memory objects returned.

SVCDUMPRGN=NO
The memory objects with the SVCDUMPRGN=NO attribute are included in the set of memory objects returned.

SVCDUMPRGN=ALL
All memory objects are included in the set of memory objects returned regardless if they have the SVCDUMPRGN=YES or SVCDUMPRGN=NO attributes.

DUMP=ALL
DUMP=LIKECSA
DUMP=LIKESQA
DUMP=LIKERGN
DUMPPROTOKOL=NO
ORDER=ASCENDING
ORDER=DUMPPRIORITY
DUMPPROTOKOL=YES
When V64SELECT=YES is specified, an optional parameter that specifies whether the memory object should be included within the set of memory object descriptions returned by the LIST request.

DUMP=ALL
All memory objects, (regardless of what the SVCDUMP specification was on the IARV64 GETSTOR/GETCOMMON/SHAREMEMOBJ request) are included in the set of memory objects returned.

DUMP=LIKECSA
The 64-bit common memory objects that have the DUMP=LIKECSA attribute specified or defaulted to on the IARV64 GETCOMMON request are included in the set of memory objects returned.

DUMP=LIKESQA
The 64-bit common memory objects that have the DUMP=LIKESQA attribute specified or defaulted to on the IARV64 GETCOMMON request are included in the set of memory objects returned.

DUMP=LIKERGN
The 64-bit private or 64-bit shared memory objects that have the
DUMP=LIKEGRN attribute specified or defaulted to on the IARV64 GETSTOR/SHAREMEMOBJ request are included in the set of memory objects returned.

,DUMPPROTOCOL=NO
,DUMPPROTOCOL=YES

An optional input parameter that specifies whether or not special selection criteria should be applied to the set of memory object descriptions returned by the LIST request. The DEFAULT is DUMPPROTOCOL=NO.

,DUMPPROTOCOL=NO
No additional selection criteria is applied.

,DUMPPROTOCOL=YES
When USERTOKEN=usertoken and SVCDUMPRGN=YES are specified, memory objects are returned according to certain selection criteria.

,ORDER=ASCENDING
,ORDER=DUMPPRIORITY

When V64SELECT=YES and DUMPPROTOCOL=NO is specified, an optional parameter that specifies the order in which the memory objects matching the selection criteria on the LIST request will be returned. The default is ORDER=ASCENDING.

,ORDER=ASCENDING
Memory objects that match the selection criteria are returned in ascending start address order.

,ORDER=DUMPPRIORITY
Memory objects that match the selection criteria are returned in dump priority order where memory objects with higher priority are listed before memory objects with lower priority. Within a dump priority level, memory objects will be listed based on ascending start address.
ORDER=DUMPPRIORITY cannot be specified with V64SHARED=YES.
ORDER=DUMPPRIORITY also can not be specified when SVCDUMPRGN=NO is specified.

,OWNERCOM=ALL
,OWNERCOM=HOME
,OWNERCOM=PRIMARY
,OWNERCOM=SYSTEM
,OWNERCOM=BYASID

An optional keyword input that specifies the owning entity of the 64-bit common memory objects to be included in the set returned.

,OWNERCOM=ALL
The 64-bit common memory objects belonging to all ASIDs are included in the set returned.

,OWNERCOM=HOME
The 64-bit common memory objects belonging to the HOME asid are included in the set returned.

,OWNERCOM=PRIMARY
The 64-bit common memory objects belonging to the PRIMARY asid are included in the set returned.

,OWNERCOM=SYSTEM
The 64-bit common memory objects belonging to the SYSTEM (not associated with an address space) are included in the set returned.
The 64-bit common memory objects belonging to a specific ASID are included in the set returned.

The name of an optional halfword integer input specifying the owning ASID of the 64-bit common memory objects to be included in the set returned.

An optional input parameter that specifies which memory objects should be included within the set of memory object descriptions returned by the LIST request. The DEFAULT is PAGEFRAMESIZE=ALL

All memory objects are included in the set of memory objects returned regardless of the page frame size.

The memory objects which were backed by 4 KB frames are included in the set of memory objects returned.

The memory objects which were backed by 1MEG frames are included in the set of memory objects returned.

When V64SELECT=YES and V64COMMON=YES are specified, an optional input parameter specifying the owning jobname of the 64-bit common memory objects to be included in the set returned by the LIST request. The default is ALL.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

The following ATTRIBUTE options are used to determine which memory objects should be included within the set of memory object descriptions returned by the LIST request: NOTOWNERGONE, OWNERGONE.

One or more values may be specified for the ATTRIBUTE parameter. If more than one value is specified, group the values within parentheses.
IARV64 macro

\texttt{,RETCODE=retcode}

An optional output parameter into which the return code is to be copied from GPR 15.

\textbf{To code:} Specify the RS-type address of a fullword field, or register (2)-(12).

\texttt{,RSNCODE=rsncode}

An optional output parameter into which the reason code is to be copied from GPR 0.

\textbf{To code:} Specify the RS-type address of a fullword field, or register (2)-(12).

\texttt{,PLISTVER=IMPLIED\_VERSION}

\texttt{,PLISTVER=MAX}

\texttt{,PLISTVER=0, 1, 2, 3, 4, 5g}

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- \textbf{IMPLIED\_VERSION}, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED\_VERSION is the default.

- \textbf{MAX}, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- \textbf{0}, supports all parameters except those specifically referenced in higher versions.

- \textbf{1}, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED

- \textbf{2}, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALLYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
IARV64 macro

- OWNERCOM
- TYPE
- UNLOCKED
- USERTOKEN
- V64COMMON

- 3, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO

- 4, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

- 5, supports both the following parameters and parameters from versions 0, 1, 2, 3, 4:
  - UNITS
  - UNITSIZE

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4

, MF=S
, MF=(L, list addr)
, MF=(L, list addr, attr)
, MF=(L, list addr, 0D)
, MF=(E, list addr)
, MF=(E, list addr, COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

, list addr
  The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

, attr
  An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter
REQUEST=DETACH option of IARV64

REQUEST=DETACH allows you to free one or more memory objects.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and PSW key 8-15.</td>
</tr>
</tbody>
</table>

The caller must be running in supervisor state or with PSW key 0-7 to use the following parameters:

- AFFINITY=SYSTEM
- OWNER=NO

<table>
<thead>
<tr>
<th>Dispatchable unit mode:</th>
<th>Task or SRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
</tbody>
</table>

Note: Note that problem state caller running in PSW key 8-15 can use DETACH only when the primary address space is the home address space.

<table>
<thead>
<tr>
<th>AMODE:</th>
<th>31- or 64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>A local lock may be held, subject to the following limitation:</td>
</tr>
</tbody>
</table>

When a local lock is held for a request for non-shared memory objects, the lock must be for the address space specified (or defaulted) by the input ALETVALUE.

| Control parameters: | Control parameters must be in the primary address space and can reside both below and above the bar. |

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register

Contents
IARV64 macro

0  Reason code, if GPR 15 is non-zero
1  Used as a work register by the system
2-13 Unchanged
14  Used as a work register by the system
15  Return code

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None

Syntax
The REQUEST=DETACH option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=DETACH</td>
<td></td>
</tr>
<tr>
<td>_MATCH=SINGLE</td>
<td>Default: MATCH=SINGLE</td>
</tr>
<tr>
<td>_MATCH=MOTOKEN</td>
<td></td>
</tr>
<tr>
<td>_MATCH=USERTOKEN</td>
<td></td>
</tr>
<tr>
<td>_MEMOBJSTART=memobjstart</td>
<td>memobjstart: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_MOTKN=motkn</td>
<td>motkn: RS-type address or address in register (2) - (12).</td>
</tr>
</tbody>
</table>
## IARV64 macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.USERTKN=usertkn</code></td>
<td>usertkn: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.USERTKN=NO_USERTKN</code></td>
<td>Default: USERTKN=NO_USERTKN</td>
</tr>
<tr>
<td><code>.MOTKNCREATOR=USER</code></td>
<td>Default: MOTKNCREATOR=USER</td>
</tr>
<tr>
<td><code>.MOTKNCREATOR=SYSTEM</code></td>
<td>Default: MOTKNCREATOR=SYSTEM</td>
</tr>
<tr>
<td><code>.AFFINITY=LOCAL</code></td>
<td>Default: AFFINITY=LOCAL</td>
</tr>
<tr>
<td><code>.AFFINITY=SYSTEM</code></td>
<td>Default: AFFINITY=SYSTEM</td>
</tr>
<tr>
<td><code>.OWNER=YES</code></td>
<td>Default: OWNER=YES</td>
</tr>
<tr>
<td><code>.OWNER=NO</code></td>
<td>Default: OWNER=NO</td>
</tr>
<tr>
<td><code>.TTOKEN=ttoken</code></td>
<td>ttoken: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.TTOKEN=NO_TTOKEN</code></td>
<td>Default: TTOKEN=NO_TTOKEN</td>
</tr>
<tr>
<td><code>.V64COMMON=NO</code></td>
<td>Default: V64COMMON=NO</td>
</tr>
<tr>
<td><code>.V64COMMON=YES</code></td>
<td>Default: V64COMMON=YES</td>
</tr>
<tr>
<td><code>.ALETVALUE=aletvalue</code></td>
<td>aletvalue: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.ALETVALUE=0</code></td>
<td>Default: ALETVALUE=0</td>
</tr>
<tr>
<td><code>.COND=NO</code></td>
<td>Default: COND=NO</td>
</tr>
<tr>
<td><code>.COND=YES</code></td>
<td>Default: COND=YES</td>
</tr>
<tr>
<td><code>.RETCODE=retcode</code></td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>.RSNCODE=rsncode</code></td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>.PLISTVER=IMPLIED_VERSION</code></td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td><code>.PLISTVER=MAX</code></td>
<td>Default: PLISTVER=MAX</td>
</tr>
<tr>
<td><code>.PLISTVER=0, 1, 2, 3 or 4</code></td>
<td>Default: PLISTVER=0, 1, 2, 3 or 4</td>
</tr>
<tr>
<td><code>.MF=S</code></td>
<td>Default: MF=S</td>
</tr>
<tr>
<td><code>.MF=(L,list addr)</code></td>
<td>list addr: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td><code>.MF=(L,list addr,attr)</code></td>
<td>Default: MF=(L,list addr,attr)</td>
</tr>
<tr>
<td><code>.MF=(L,list addr,0D)</code></td>
<td>Default: MF=(L,list addr,0D)</td>
</tr>
<tr>
<td><code>.MF=(E,list addr)</code></td>
<td>Default: MF=(E,list addr)</td>
</tr>
<tr>
<td><code>.MF=(E,list addr,COMPLETE)</code></td>
<td>Default: MF=(E,list addr,COMPLETE)</td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

**name**

An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

**REQUEST=DETACH**

A required parameter. REQUEST=DETACH frees one or more memory objects. Problem state programs running in PSW key (8-15) can use this function only when the primary address space is the home address space, and can affect only a memory object that is created using GETSTOR CONTROL=UNAUTH. If a problem state program running in PSW key (8–15) tries to free a memory object created with CONTROL=AUTH, the system does not free the memory object and an ABEND will be issued.

A memory object can be affected by DETACH when MATCH=SINGLE is specified without MOTKN/USERTKN. Other invocations of DETACH will affect nonshared memory objects only when a matching user token is passed.

A shared memory object can be affected by DETACH only when a matching user token is passed.

When DETACH MATCH=SINGLE AFFINITY=LOCAL USERTKN is specified against a shared memory object, the shared interest will be removed from the address space designated by ALETVALUE provided the usertoken passed still represents current shared interest by the space.

1. If this address space has no further shared interest in the memory object, then DETACH will also remove addressability for the address space identified by the input ALETVALUE.

2. When the last address space has surrendered its use of a given shared memory object and the system interest has been removed (through DETACH AFFINITY=SYSTEM) the memory object will be freed.

When DETACH MATCH=USERTOKEN AFFINITY=LOCAL is specified and the input user token matches the usertoken provided for a given memory object created through GETSTOR MOTKN, that memory object is freed. If the memory object was created through GETSHARED and the input user token represents current shared interest by the address space, then that interest will be removed. The same two observations as in the prior list apply.

When DETACH MATCH=USERTOKEN AFFINITY=SYSTEM is specified, only shared memory objects are affected. When the input user token matches the system interest, the system interest will be removed. If there is no remaining local interest, then the shared memory object is freed.

All I/O into each memory object specified must be complete before the DETACH is requested. If the DETACH service finds a PAGEFIXed page in the memory object, the memory object will not be freed, but any prior pages will have indeterminate data and the caller will be abnormally ended.

**MATCH=SINGLE**

An optional parameter that indicates which memory objects are to be freed. The default is MATCH=SINGLE.
IARV64 macro

,MATCH=SINGLE
   Specifies that the input contains MEMOBJSTART for a single memory object.

,MATCH=MOTOKEN
   Specifies that the input contains a memory object token that was passed to GETSTOR, GETSHARED or SHAREMEMOBJ. Memory objects not associated with a memory object token are not affected. Such objects would have to have been created using GETSTOR without MOTKN/USERTKN. If MATCH=MOTOKEN or MATCH =USERTOKEN, COND=YES, and no matching memory object token exists, the system returns a return code instead of abnormally ending the caller.

For nonshared memory objects, all memory objects associated with this memory object token are freed unless it is a problem state program with PSW key 8-15 trying to free a memory object created with CONTROL=AUTH.

For shared memory objects, when AFFINITY=LOCAL is given, the shared interest in memory objects associated with this memory object token is to be removed (for the ALET specified through ALETVALUE). If a given shared memory object no longer has outstanding shared interest then it will be freed.

For shared memory objects, when AFFINITY=SYSTEM is given, the system interest in memory objects associated with this memory object token is to be freed. If a specified shared memory object no longer has outstanding shared interest then it will be freed.

If the system encounters an error in processing a qualifying memory object, for example, an unexpected pagefixed page, then the processing ends. The system does not process that page or any further pages or memory objects and abnormally ends the caller.

,MATCH=USERTOKEN
   This is a synonym for MATCH=MOTOKEN.

,MEMOBJSTART=memobjstart
   When MATCH=SINGLE is specified, a required input parameter that contains the address of the first byte in the memory object.

   To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,MOTKN=motoken
   A optional input parameter that identifies the memory object token to uniquely identify the memory object, as previously passed to GETSTOR, GETSHARED or SHAREMEMOBJ, or the token that was generated by the system on a GETCOMMON or GETSTOR request.

   Each shared memory object can be associated with multiple memory object tokens. For AFFINITY=LOCAL, the shared interest in a shared memory object associated with this memory object token is to be removed for the address space. For AFFINITY=SYSTEM, the shared interest created by GETSHARED is to be removed or the 64-bit common memory object is to be freed. For either specification of AFFINITY, when a given shared memory object no longer has outstanding shared interest, it is freed.

   When the memory object is not associated with the input token value, it is not processed.
To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field, or specify a literal decimal value.

,USERTKN=usertkn
USERTKN=NO_USERTKN
A parameter that is similar to MOTKN. Unlike MOTKN, a USERTKN is always presumed to be user-created. The default is NO_USERTKN. It is suggested that you use MOTKN rather than USERTKN.
MOTKN=xxx,MOTKNCREATOR=USER is equivalent to USERTKN=xxx.

The default can be used only for memory objects created by GETSTOR. When the memory object is created by GETSHARED, it is necessary to specify the memory object token to uniquely identify which shared interest is to be freed.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field, or specify a literal decimal value.

,MOTKNCREATOR=USER
MOTKNCREATOR=SYSTEM
When MOTKN=motokn is specified, an optional parameter that indicates who created the memory object token.

,MOTKNCREATOR=USER
The memory object token is user-created.

,MOTKNCREATOR=SYSTEM
The memory object token is system-created.

The default is MOTKNCREATOR=USER.

,AFFINITY=LOCAL
,AFFINITY=SYSTEM
An optional input parameter that identifies whether local or system affinity for the memory object will be affected. The default is AFFINITY=LOCAL.

,AFFINITY=LOCAL
Local affinity to the memory object is to be affected, the interest in the memory object defined by the input ALETVALUE and memory object token. Nonshared memory objects are affected by AFFINITY=LOCAL.

Shared memory objects for which an appropriate SHAREMEMOBJ has been done by the address space defined by the input ALETVALUE will also by affected by AFFINITY=LOCAL.

64-Bit Common memory objects are not affected by AFFINITY=LOCAL.

AFFINITY=SYSTEM
System affinity to the shared or 64-bit common memory object will be affected.

AFFINITY=SYSTEM can be used only by callers running in supervisor state of with PSW key 0–7.

,OWNER=YES
,OWNER=NO
When AFFINITY=LOCAL is specified, an optional keyword input that specifies whether the system will check if the ttoken provided or the task of the caller matches the ttoken associated with the memory object when it was created (only relevant for memory objects created through GETSTOR not GETSHARED). The default is OWNER=YES.
The task which owns the memory object must match the current task or the ttoken provided.

The task which is freeing the memory object does not have to be the owner of the memory object. NO can be used only by programs running in supervisor state or with PSW key 0-7.

When OWNER=YES and AFFINITY=LOCAL are specified, an optional input parameter that identifies the task that owns the memory object. The TTOKEN is returned by the TCBTOKEN macro.

If TTOKEN is not specified, the task issuing the DETACH request must be the owner of the memory object.

The task identified by the TTOKEN must be in the address space specified or defaulted by the ALETVALUE keyword.

When the TTOKEN parameter is used by problem state programs with PSW key 8-15, the target task must represent the calling task OR the jobstep task for the calling task OR the mother task. The mother task may not be given however when the calling task is itself a jobstep task.

If the TTOKEN parameter is not specified, and the caller is a TCB, the currently dispatched task must be the owner of the memory object. When OWNER= YES is specified by an SRB, the caller will be abnormally ended if the TTOKEN value is not supplied. The default is NO_TTOKEN.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

When AFFINITY=SYSTEM is specified, an optional input parameter that indicates whether this is memory object is a 64-bit common memory object. The default is V64COMMON=NO.

This is not a 64-bit common memory object.

This is a 64-bit common memory object.

An optional input parameter that indicates the ALET of the address space of the memory object to be freed.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by programs running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

An optional keyword input that specifies whether the request is unconditional or conditional. When you code COND=YES and there is insufficient storage to satisfy the request, instead of the request being abnormally ended the request
IARV64 macro

The request is unconditional. The request will be abnormally ended when the request cannot be satisfied.

,COND=YES

The request is conditional. The request will not be abnormally ended for resource unavailability.

,RETCODE=retcode

An optional output parameter into which the return code is to be copied from GPR 15. If you specify register 15, the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or register (15).

,RSNCODE=rsncode

An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
IARV64 macro

- DETACHFIXED
- DOAUTHCHECKS
- DUMP
- DUMPPRIORITY
- DUMPPROTOCOL
- LOCALSYSAREA
- MEMLIMIT
- OPTIONVALUE
- ORDER
- OWNERASID
- OWNERCOM
- TYPE
- UNLOCKED
- USERTOKEN
- V64COMMON

• 3, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO

• 4, supports both the following parameters and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:

• IMPLIED_VERSION
• MAX
• A decimal value of 0, 1, 2, 3 or 4

, MF=S
, MF=(L, list addr)
, MF=(L, list addr, attr)
, MF=(L, list addr, 0D)
, MF=(E, list addr)
, MF=(E, list addr, COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.
IARV64 macro

,,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

,,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=GETSHAREO option of IARV64
REQUEST=GETSHARE creates a memory object that can be shared across multiple address spaces.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>The caller must be running in supervisor state or with PSW key 0-7 to use the following parameters.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN.</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks may be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space and can reside both below and above the bar.</td>
</tr>
</tbody>
</table>

Programming requirements
None

Restrictions
This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information
Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>
**IARV64 macro**

2-13  Unchanged  
14   Used as a work register by the system  
15   Return code

When control returns to the caller, the ARs contain:

Register Contents

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None

**Syntax**

The REQUEST=GETSHARED option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=GETSHARED</td>
<td></td>
</tr>
<tr>
<td>,COND=NO</td>
<td>Default: COND=NO</td>
</tr>
<tr>
<td>,COND=YES</td>
<td></td>
</tr>
<tr>
<td>,SEGMENTS=segments</td>
<td>segments: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,KEY=key</td>
<td><em>key</em>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,KEY=CALLERKEY</td>
<td>Default: KEY=CALLERKEY</td>
</tr>
<tr>
<td>,FPROT=YES</td>
<td>Default: FPROT=YES</td>
</tr>
<tr>
<td>,FPROT=NO</td>
<td></td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=GETSHARED
REQUEST=GETSHARED requests that a memory object be created. The memory object is allowed to be shared upon return (through SHAREMEMOBJ). Successful completion of this service creates system interest in the memory object, which must be removed (through DETACH AFFINITY=SYSTEM) before the memory object is freed. Addressability to the memory object is not provided by GETSHARED. Instead, use SHAREMEMOBJ to enable the virtual storage to be referenced. A memory object created through GETSHARED is not eligible for PAGEFIX or PAGEUNFIX.

,COND=NO
An optional keyword input that specifies whether the request is unconditional or conditional. When you code COND=YES and there is insufficient storage to satisfy the request, instead of the request being abnormally ended, the request will complete but a return code will be set to indicate that the request could not be completed successfully. In all cases, the request will be abnormally ended for invalid requests, including violation of environmental restrictions. The default is COND=NO.

The request is unconditional. The request will be abnormally ended when the request cannot be satisfied.

The request is conditional. The request will not be abnormally ended for resource unavailability.

A required input parameter that specifies the size of storage requested in megabytes. This must be a non-zero value.

The amount of storage requested is not charged against the MEMLIMIT.

To code: Specify the RS-type address, or address in register (2)-(12), of a required doubleword field.

An optional input parameter that specifies the storage key to be assigned to the memory object. The key must be in bits 0-3 of the specified byte. Bits 4-7 are ignored.

If the key is not specified, the storage key of the memory object is the same as the caller's PSW key. The default is KEY=CALLERKEY.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

An optional parameter that specifies whether the memory object should be fetch protected. The default is FPROT=YES.

The entire memory object will be fetch protected. A program must have a PSW key that matches the storage key of the memory object (or have PSW key 0) to reference data in the memory object.

The memory object will not be fetch protected.

This parameter is the name (RS-type), or address in register (2)-(12), of a required doubleword input that identifies the user token to be associated with the shared memory object. This can be used on a later DETACH invocation to affect all memory objects associated with this value. A single shared memory object may be associated with multiple user tokens via GETSHARED and SHAREMEMOBJ.

CHANGEACCESS=LOCAL
\textbf{CHANGEACCESS=GLOBAL}

An optional parameter that specifies whether the subsequent
\texttt{CHANGEACCESS} requests are treated as local or global. The default is
\texttt{CHANGEACCESS=LOCAL}.

\textbf{CHANGEACCESS=LOCAL}

The \texttt{CHANGEACCESS} for this memory object will have local scope.
Subsequent \texttt{CHANGEACCESS} requests will change access only for the
address space specified by \texttt{CHANGEACCESS}.

\textbf{CHANGEACCESS=GLOBAL}

The \texttt{CHANGEACCESS} for this memory object will have global scope.
Subsequent \texttt{CHANGEACCESS} requests will change access for all address
spaces sharing the memory object and any new address spaces that will
subsequently share it.

\textbf{Note:} Use of \texttt{GLOBAL} may reduce system resources needed to manage the
memory object and is encouraged when all spaces will be using the same view.

\textbf{ALETVALUE=\texttt{aletvalue}}

\textbf{ALETVALUE=0}

This parameter is the name of an optional fullword integer input that indicates
the ALET of the address space which will be used to create the memory object.
The only supported values are 0 (primary) and 2 (home). The default value is 0.

\textbf{ORIGIN=\texttt{origin}}

A required output parameter that contains the lowest address of the memory
object.

\textbf{To code:} Specify the RS-type address, or address in register (2)-(12), of an
eight-byte pointer field.

\textbf{RETCODE=\texttt{retcode}}

An optional output parameter into which the return code is to be copied from
GPR 15.

\textbf{To code:} Specify the RS-type address of a fullword field, or register (2)-(12).

\textbf{RSNCODE=\texttt{rsncode}}

An optional output parameter into which the reason code is to be copied from
GPR 0.

\textbf{To code:} Specify the RS-type address of a fullword field, or register (2)-(12).

\textbf{PLISTVER=IMPLIED\_VERSION}

\textbf{PLISTVER=MAX}

\textbf{PLISTVER=0, 1, 2, 3 or 4}

An optional input parameter that specifies the version of the macro. PLISTVER
determines which parameter list the system generates. PLISTVER is an
optional input parameter on all forms of the macro, including the list form.
When using PLISTVER, specify it on all macro forms used for a request and
with the same value on all of the macro forms. The values are:

\begin{itemize}
  \item \texttt{IMPLIED\_VERSION}, which is the lowest version that allows all parameters
  specified on the request to be processed. If you omit the PLISTVER
  parameter, IMPLIED\_VERSION is the default.
  \item \texttt{MAX}, if you want the parameter list to be the largest size currently possible.
  This size might grow from release to release and affect the amount of
  storage that your program needs.
\end{itemize}
IARV64 macro

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- 0, supports all parameters except those specifically referenced in higher versions.
- 1, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- 2, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERSID
  - OWNERCMD
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- 3, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- 4, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3 or 4

`MF=S`  
`MF=(L,list addr)`  
`MF=(L,list addr,attr)`  
`MF=(L,list addr,0D)`
IARV64 macro

, MF=(E, list addr)
, MF=(E, list addr, COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
   The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
   An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

,COMPLETE
   Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=SHAREMEMOBJ option of IARV64

REQUEST=SHAREMEMOBJ requests that the address space be given access to one or more specified shared memory objects.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>The caller must be running in supervisor state or with PSW key 0-7.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN.</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>When a local lock is held for a request the lock must be for the address space specified (or defaulted) by the input ALETVALUE.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space and can reside both below and above the bar.</td>
</tr>
</tbody>
</table>

Programming requirements

None.
Restrictions
This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information
Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None

Syntax
The REQUEST=SHAREMEMOBJ option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
</tbody>
</table>
# Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST=SHAREMEMOBJ</td>
<td></td>
</tr>
<tr>
<td>_USERTKN=usertkn</td>
<td>usertkn: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_RANGLIST=ranglist</td>
<td>ranglist: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_NUMRANGE=numrange</td>
<td>numrange: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_NUMRANGE=1</td>
<td>Default: NUMRANGE=1</td>
</tr>
<tr>
<td>_ALETVALUE=aletovalue</td>
<td>aletvalue: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_ALETVALUE=0</td>
<td>Default: ALETVALUE=0</td>
</tr>
<tr>
<td>_SVCDUMPRGN=YES</td>
<td>Default: SVCDUMPRGN=YES</td>
</tr>
<tr>
<td>_DUMPPRIORITY=dumppriority</td>
<td></td>
</tr>
<tr>
<td>_SVCDUMPRGN=NO</td>
<td></td>
</tr>
<tr>
<td>_COND=NO</td>
<td>Default: COND=NO</td>
</tr>
<tr>
<td>_COND=YES</td>
<td></td>
</tr>
<tr>
<td>_RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>_PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>_PLISTVER=0, 1, 2, 3 or 4</td>
<td></td>
</tr>
<tr>
<td>_MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>_MF=(L, list addr)</td>
<td>list addr: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td>_MF=(L, list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>_MF=(L, list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>_MF=(E, list addr)</td>
<td></td>
</tr>
<tr>
<td>_MF=(E, list addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

## Parameters
The parameters are explained as follows:
IARV64 macro

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=SHAREMEMOBJ
REQUEST=SHAREMEMOBJ requests that the caller wants to be given shared access to the specified memory object. The memory object specified must be a SHARED memory object, such as the result of a GETSHARED invocation.

,USERTKN=usertkn
A required doubleword input parameter that identifies the user token to be associated with the memory object. This can be used on a later DETACH request to free all memory objects associated with this value.

To avoid inadvertent collisions in the values specified, the left word (bits 0-31) should represent an address of some storage related to the caller. The system enforces the rule that the left word is non-zero for authorized callers. The right word should represent the virtual address of some storage related to the caller, which could be a control block address, an entry point address, etc.; the choice of which to use is made by the application.

To code: Specify the RS-type address, or address in register (2)-(12), of a required doubleword field that identifies the user token to be associated with the shared memory object.

,RANGLIST=ranglist
A required input parameter. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

VSA
VSA denotes the starting address of the data to be acted on. The virtual address must be within a memory object returned by GETSHARED (not GETSTOR or GETCOMMON).

The length of this field is 8 bytes.

RESERVED
Reserved for future use, must be in binary zeros.

The length of this field is 8 bytes.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,NUMRANGE=numrange

,NUMRANGE=1
An optional input parameter that specifies the number of entries in the supplied range list. The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,ALETVALUE=aletvalue

,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space that will be given access to the shared memory object.
The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a fullword field.

- **SVCDDUMPREGN=YES**
- **SVCDDUMPREGN=NO**

An optional parameter that specifies whether the memory object should be included in an SVC dump when region is requested. The default is SVCDDUMPREGN=YES.

- **SVCDDUMPREGN=NO**

The memory object should not be included in an SVC dump when RGN is specified on SDATA.

- **DUMPPRIORITY=dumppriority**

The name of an optional one-byte integer input parameter that specifies the dump priority of the memory object. This must be a non-zero value in the range of 1-99, with 1 being the highest priority and 99 being the lowest. The default value is 99.

- **SVCDDUMPREGN=NO**

The memory object should not be included in an SVC dump when RGN is specified on SDATA.

- **COND=NO**
- **COND=YES**

An optional keyword input that specifies whether the request is unconditional or conditional. When you code COND=YES and there is insufficient storage to satisfy the request, instead of the request being abnormally ended the request will complete but a return code will be set to indicate that the request could not be completed successfully. In all cases the request will be abnormally ended for invalid requests, including violation of environmental restrictions. The default is COND=NO.

- **COND=NO**

The request is unconditional. The request will be abnormally ended when the request cannot be satisfied.

- **COND=YES**

The request is conditional. The request will not be abnormally ended for resource unavailability.

- **RETCODE=retcode**

An optional output parameter into which the return code is to be copied from GPR 15.

  **To code:** Specify the RS-type address of a fullword field, or register (2)-(12).

- **RSNCODE=rsncode**

An optional output parameter into which the reason code is to be copied from GPR 0.

  **To code:** Specify the RS-type address of a fullword field, or register (2)-(12).

- **PLISTVER=IMPLIED_VERSION**
- **PLISTVER=MAX**
- **PLISTVER=0, 1, 2, 3 or 4**

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an
optional input parameter on all forms of the macro, including the list form.
When using PLISTVER, specify it on all macro forms used for a request and
with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters
  specified on the request to be processed. If you omit the PLISTVER
  parameter, IMPLIED_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible.
  This size might grow from release to release and affect the amount of
  storage that your program needs.
  
  If you can tolerate the size change, IBM recommends that you always
  specify PLISTVER=MAX on the list form of the macro. Specifying MAX
  ensures that the list-form parameter list is always long enough to hold all
  the parameters you might specify on the execute form, when both are
  assembled with the same level of the system. In this way, MAX ensures that
  the parameter list does not overwrite nearby storage.

- **0**, supports all parameters except those specifically referenced in higher
  versions.

- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED

- **2**, supports both the following parameters and parameters from version 0
  and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERSID
  - ORNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON

- **3**, supports both the following parameters and parameters from versions 0,
  1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO

- **4**, supports both the following parameter and parameters from versions 0, 1,
  2, 3:
  - DMAPAGETABLE
To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3 or 4

\( \text{MF=S} \)
\( \text{MF=(L,list addr)} \)
\( \text{MF=(L,list addr,attr)} \)
\( \text{MF=(L,list addr,OD)} \)
\( \text{MF=(E,list addr)} \)
\( \text{MF=(E,list addr,COMPLETE)} \)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

\( \text{list addr} \)
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

\( \text{attr} \)
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code \text{attr}, the system provides a value of 0D.

\( \text{COMPLETE} \)
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

**REQUEST=CHANGEACCESS option of IARV64**

REQUEST=CHANGEACCESS requests that the view type for segments within the specified 64-bit shared memory objects can be changed.

**Environment**

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>The caller must be running in supervisor state or with PSW key 0-7.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN.</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
</tbody>
</table>
IARV64 macro

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks may be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space and can reside both below and above the bar.</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=CHANGEACCESS option of the IARV64 macro is written as follows:
### IARV64 macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td>IARV64</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must follow IARV64.</td>
</tr>
</tbody>
</table>

REQUEST=CHANGEACCESS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.VIEW=READONLY</code></td>
<td>Default: VIEW=READONLY</td>
</tr>
<tr>
<td><code>.VIEW=SHAREDWRITE</code></td>
<td></td>
</tr>
<tr>
<td><code>.VIEW=HIDDEN</code></td>
<td></td>
</tr>
<tr>
<td><code>.RANGLIST=ranglist</code></td>
<td><code>ranglist</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.NUMRANGE=numrange</code></td>
<td><code>numrange</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.NUMRANGE=1</code></td>
<td>Default: NUMRANGE=1</td>
</tr>
<tr>
<td><code>.ALETVALUE=aletvalue</code></td>
<td><code>aletvalue</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.ALETVALUE=0</code></td>
<td>Default: ALETVALUE=0</td>
</tr>
<tr>
<td><code>.RETCODE=retcode</code></td>
<td><code>retcode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>.RSNCODE=rsncode</code></td>
<td><code>rsncode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>.PLISTVER=IMPLIED_VERSION</code></td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td><code>.PLISTVER=MAX</code></td>
<td></td>
</tr>
<tr>
<td><code>.PLISTVER=0, 1, 2, 3 or 4</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=S</code></td>
<td>Default: MF=S</td>
</tr>
<tr>
<td><code>.MF=(L,list addr)</code></td>
<td><code>list addr</code>: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td><code>.MF=(L,list addr,attr)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(L,list addr,0D)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(E,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(E,list addr,COMPLETE)</code></td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:
IARV64 REQUEST=CHANGEACCESS requests that the view type for segments within the specified 64-bit shared memory objects can be changed.

\textit{name}

An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

\textbf{REQUEST=CHANGEACCESS}

REQUEST=CHANGEACCESS requests that the type of access to the specified virtual storage be changed. For 64-bit shared memory objects the scope of the change is determined by the choice of LOCAL versus GLOBAL on the IARV64 GETSHARED CHANGEACCESS keyword.

For 64-bit shared memory objects when CHANGEACCESS=LOCAL was specified or defaulted on the REQUEST=GETSHARED, only the address space specified by the ALET is affected.

For 64-bit shared memory objects when the CHANGEACCESS=GLOBAL is specified, all address spaces currently sharing the memory object are affected. Subsequent IARV64 SHAREMEMOBJ requests for this memory object will also be affected by this CHANGEACCESS when CHANGEACCESS=GLOBAL is specified (until the next CHANGEACCESS invocation).

The memory object specified must be a 64-bit shared memory object. For example, it is the result of a GETSHARED invocation.

CHANGEACCESS requests for memory objects that are CHANGEACCESS=LOCAL require that the target space have interest in the shared memory object. For example, a SHAREMEMOBJ for the target space must have been done before the CHANGEACCESS request. Memory objects with CHANGEACCESS=GLOBAL support CHANGEACCESS requests without prior SHAREMEMOBJ requests.

\texttt{,VIEW=READONLY}

\texttt{,VIEW=SHAREDWRITE}

\texttt{,VIEW=HIDDEN}

A required input parameter that indicates the accessing mode on the area.

\texttt{,VIEW=READONLY}

This parameter specifies that the area can only be used to read data. Any attempt to alter data by writing onto the area will result in a program check.

\texttt{,VIEW=SHAREDWRITE}

This parameter specifies that the area can be used to read or update data.

\texttt{,VIEW=HIDDEN}

This parameter specifies that the data within the area cannot be accessed until its view type is changed to READONLY or SHAREDWRITE. Any attempt to access a hidden area will result in a program check.

\texttt{,RANGLIST=ranglist}

A required input parameter that contains the address of the ranglist. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each range list entry follows:

\texttt{VSA}

VSA denotes the starting virtual address of the data to be acted on. The
IARV64 macro

virtual address specified must be within a memory object returned by
GETSHARED (not GETSTOR or GETCOMMON). The value must always
be on a segment boundary.

NUMSEGMENTS

NUMSEGMENTS contains the number of segments (megabytes) in the
area. The number of segments specified starting with the specified VSA
must lie within a single memory object. The length of this field is 8 bytes.

To code: Specify the RS-type address, or address in register (2)-(12), of an
eight-byte pointer field.

,NUMRANGE=numrange

,NUMRANGE=1

An optional input parameter that specifies the number of entries in the
supplied range list.

The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a
fullword field.

,ALETVALUE=aletvalue

,ALETVALUE=0

An optional input parameter that indicates the ALET of the address space
sharing a memory object that will change access to the memory object.

The only supported values are 0 (primary) and 2 (home).

To code: Specify the RS-type address, or address in register (2)-(12), of a
fullword field.

,RETCODE=retcode

An optional output parameter into which the return code is to be copied from
GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=rsncode

An optional output parameter into which the reason code is to be copied from
GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION

,PLISTVER=MAX

,PLISTVER=0, 1, 2, 3 or 4

An optional input parameter that specifies the version of the macro. PLISTVER
determines which parameter list the system generates. PLISTVER is an
optional input parameter on all forms of the macro, including the list form.
When using PLISTVER, specify it on all macro forms used for a request and
with the same value on all of the macro forms. The values are:

- IMPLIED_VERSION, which is the lowest version that allows all parameters
  specified on the request to be processed. If you omit the PLISTVER
  parameter, IMPLIED_VERSION is the default.
- MAX, if you want the parameter list to be the largest size currently possible.
  This size might grow from release to release and affect the amount of
  storage that your program needs.

If you can tolerate the size change, IBM recommends that you always
specify PLISTVER=MAX on the list form of the macro. Specifying MAX
ensures that the list-form parameter list is always long enough to hold all
the parameters you might specify on the execute form, when both are
assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- 0, supports all parameters except those specifically referenced in higher versions.
- 1, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- 2, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOR
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- 3, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- 4, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3 or 4

\[\text{MF}=S\]
\[\text{MF}=(L, \text{list addr})\]
\[\text{MF}=(L, \text{list addr}, \text{attr})\]
\[\text{MF}=(L, \text{list addr}, \text{OD})\]
\[\text{MF}=(E, \text{list addr})\]
\[\text{MF}=(E, \text{list addr}, \text{COMPLETE})\]

An optional input parameter that specifies the macro form.
Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

\list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

\attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code \attr, the system provides a value of 0D.

\COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

**REQUEST=GETCOMMON option of IARV64**

Use REQUEST=GETCOMMON to create a 64-bit common memory object.

**Environment**

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>The caller must be running in supervisor state and with PSW key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts Callers that specify PAGEFRAMESIZE=1M or PAGEFRAMESIZE=MAX must be enabled</td>
</tr>
<tr>
<td>Locks</td>
<td>For enabled callers no requirement. For disabled callers no spin locks higher than the RSM locks can be held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be in the primary address space and can reside both below and above the bar.</td>
</tr>
</tbody>
</table>

**Programming requirements**

None.
IARV64 macro

Restrictions
This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information
Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
The REQUEST=GETCOMMON option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>REQUEST(=)GETCOMMON</td>
<td></td>
</tr>
<tr>
<td>,(\text{COND}=\text{NO})</td>
<td>Default: (\text{COND}=\text{NO})</td>
</tr>
<tr>
<td>,(\text{COND}=\text{YES})</td>
<td></td>
</tr>
<tr>
<td>(\text{,SEGMEN}T_\text{S}=\text{segments})</td>
<td>(\text{segments}): RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>(\text{,\text{PAGEFRAMESIZE}}=\text{4K})</td>
<td>Default: (\text{PAGEFRAMESIZE}=\text{4K})</td>
</tr>
<tr>
<td>(\text{,PAGEFRAMESIZE}=\text{1MEG})</td>
<td></td>
</tr>
<tr>
<td>(\text{,PAGEFRAMESIZE}=\text{MAX})</td>
<td></td>
</tr>
<tr>
<td>(\text{,PAGEFRAMESIZE}=\text{PAGEABLE1MEG})</td>
<td></td>
</tr>
<tr>
<td>(\text{,PAGEFRAMESIZE}=\text{DREF1MEG})</td>
<td></td>
</tr>
</tbody>
</table>
| \(\text{,TYPE}=\text{PAGEABLE}\) | Default: \(\text{TYPE}=\text{PAGEABLE}\) when one of the following is specified:  
- \(\text{PAGEFRAMESIZE}=\text{PAGEABLE1MEG}\)  
- \(\text{PAGEFRAMESIZE}=\text{4K}\)  
- \(\text{PAGEFRAMESIZE}=\text{MAX}\) and the memory object is backed with 4 KB-page frames.  
When \(\text{PAGEFRAMESIZE}=\text{DREF1MEG}\) is specified, the default value is \(\text{TYPE}=\text{DREF}\). If \(\text{PAGEFRAMESIZE}=\text{1MEG}\) or \(\text{PAGEFRAMESIZE}=\text{MAX}\) is specified and the memory object is backed with 1 MB-page frames, the 1 MB-pages backing this memory object are fixed. |
| \(\text{,TYPE}=\text{DREF}\) |  |
| \(\text{,UNITS}=\text{units}\) | \(\text{units}\): Size of the memory object, which is the number of units specified by UNITSIZE. |
| \(\text{,UNITSIZE}=\text{1M}\) | Specifies a 1 MB unit size. |
| \(\text{,PAGEFRAMESIZE}=\text{4K}\) | If UNITSIZE=1M is specified, a PAGEFRAMESIZE of 4K or 1M must be specified. There is no default value. |
| \(\text{,\text{TYPE}}=\text{FIXED}\) |  |
| \(\text{,\text{TYPE}}=\text{DREF}\) |  |
| \(\text{,\text{TYPE}}=\text{PAGEABLE}\) |  |
| \(\text{,\text{KEY}}=\text{key}\) | \(\text{key}\): RS-type address or address in register (2) - (12). |
| \(\text{,\text{KEY}}=\text{CALLERKEY}\) | Default: \(\text{KEY}=\text{CALLERKEY}\) |
| \(\text{,\text{FPROT}}=\text{YES}\) | Default: \(\text{FPROT}=\text{YES}\) |
| \(\text{,\text{FPROT}}=\text{NO}\) |  |
| \(\text{,\text{MOTKNSOURCE}}=\text{USER}\) | Default: \(\text{MOTKNSOURCE}=\text{USER}\) |
| \(\text{,\text{MOTKN}}=\text{motkn}\) | \(\text{motkn}\): RS-type address or address in register (2) - (12). |
| \(\text{,\text{MOTKNSOURCE}}=\text{SYSTEM}\) |  |
### Parameters

The REQUEST=GETCOMMON option of the IARV64 macro is written as follows:

```plaintext
name
```

An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

---

#### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.OUTMOTKN=outmotkn</code></td>
<td><code>outmotkn</code>: RS-type address or address in register (2) – (12).</td>
</tr>
<tr>
<td><code>.OWNERCOM=HOME</code></td>
<td><strong>Default</strong>: OWNER=HOME</td>
</tr>
<tr>
<td><code>.OWNERCOM=PRIMARY</code></td>
<td></td>
</tr>
<tr>
<td><code>.OWNERCOM=SYSTEM</code></td>
<td></td>
</tr>
<tr>
<td><code>.OWNERASID=ownerasid</code></td>
<td></td>
</tr>
<tr>
<td><code>.OWNERASID=HOME</code></td>
<td></td>
</tr>
<tr>
<td><code>.DUMP=LIKECSA</code></td>
<td><strong>Default</strong>: DUMP=LIKECSA when TYPE=PAGEABLE</td>
</tr>
<tr>
<td><code>.DUMP=LIKESQA</code></td>
<td><strong>Default</strong>: DUMP=LIKESQA when TYPE=DREF</td>
</tr>
<tr>
<td><code>.DUMP=NO</code></td>
<td></td>
</tr>
<tr>
<td><code>.DUMP=BYOPTIONVALUE</code></td>
<td></td>
</tr>
<tr>
<td><code>.OPTIONVALUE=optionvalue</code></td>
<td><code>option</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.DETACHFIXED=YES</code></td>
<td><strong>Default</strong>: DETACHFIXED=NO</td>
</tr>
<tr>
<td><code>.ORIGIN=origin</code></td>
<td><code>origin</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.RETCODE=retcode</code></td>
<td><code>retcode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>.RSNCODE=rsncode</code></td>
<td><code>rsncode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>.PLISTVER=IMPLIED_VER</code></td>
<td><strong>Default</strong>: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td><code>.PLISTVER=MAX</code></td>
<td></td>
</tr>
<tr>
<td><code>.PLISTVER=0, 1, 2, 3, 4, 5</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=S</code></td>
<td><strong>Default</strong>: MF=S</td>
</tr>
<tr>
<td><code>.MF=(L,list addr)</code></td>
<td>`(L,list addr): RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td><code>.MF=(L,list addr,attr)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(L,list addr,OD)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(E,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(E,list addr,COMPETE)</code></td>
<td></td>
</tr>
</tbody>
</table>
REQUEST=GETCOMMON
A required parameter. REQUEST=GETCOMMON creates a 64-bit common memory object.

,COND=NO
,COND=YES
An optional keyword input that specifies whether the request is unconditional or conditional. When you code COND=YES and there is insufficient storage to satisfy the request, instead of the request being abnormally ended, the request will complete and a return code will be set to indicate that the request could not be completed successfully. In all cases the request will be abnormally ended for invalid requests, including violation of environmental restrictions. The default is COND=NO.

,COND=NO
The request is unconditional. The request will be abnormally ended when the request cannot be satisfied.

,COND=YES
The request is conditional. The request will not be abnormally ended for resource unavailability.

,KEY=key
,KEY=CALLERKEY
An optional input parameter that specifies the storage key to be assigned to the memory object. The key must be in bits 0-3 of the specified byte. Bits 4-7 are ignored. Only keys 0-7 can be specified.

If the key is not specified, the storage key of the memory object is the same as the caller's PSW key. The default is CALLERKEY.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,FPROT=NO
,FPROT=YES
An optional keyword parameter that specifies whether the memory object should be fetch protected. The default is FPROT=YES

,FPROT=YES
The entire memory object will be fetch protected. A program must have a PSW key that matches the storage key of the memory object (or have PSW key 0) to reference data in the memory object.

,FPROT=NO
The memory object will not be fetch protected.

,MOTKNSOURCE=USER
,MOTKNSOURCE=SYSTEM
An optional input parameter that indicates the source of the memory object token to be associated with this memory object. The default is USER.

,MOTKNSOURCE=USER
The user provides the memory object token.

,MOTKN=motkn
The name of an optional doubleword integer input that identifies the token to be associated with the memory object. This must be a token that was returned by the system on a previous GETCOMMON request by the OUTMOTKN keyword. If you specify no user token, the default is that no user token is supplied to associate this memory object with others.
The system provides the memory object token.

OUTMOTKN=xoutmotkn

The name of a required doubleword integer output in which the system returns the token associated with this memory object. This token can be used on subsequent GETCOMMON requests as a user-supplied token in order to associate other memory objects with this token. This token can be used on subsequent DETACH requests in order to free all the memory objects that have been associated with this token.

Usage notes of the MOTKNSOURCE parameter on an IARV64 REQUEST(GETCOMMON) request:

- If you want a system-generated token to be returned, invoke:
  1ARV64 REQUEST=GETCOMMON,MOTKNSOURCE=SYSTEM,OUTMOTKN=mytoken

- If you want to use the returned token on subsequent IARV64 GETCOMMON requests in order to associate other memory objects with the same token, invoke:
  1ARV64 REQUEST=GETCOMMON,MOTKNSOURCE=USER,MOTKN=mytoken

- If you want to use the returned token on a DETACH request in order to detach all memory objects that are associated with that token, invoke:
  1ARV64 REQUEST=DETACH,MATCH=MOTOKEN,MOTKN=mytoken,
  AFFINITY=SYSTEM,V64COMMON=YES

SEGMENTS=segments

SEGMENTS and UNITS are mutually exclusive keys. This set is required; only one key can be specified.

A required input parameter that specifies the size of the memory object requested, in megabytes. This must be a nonzero value. The amount of storage requested that is not in the guard state is charged against the MEMLIMIT for the address space where the memory object is to be created.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

PAGEFRAMESIZE=4K
PAGEFRAMESIZE=1MEG
PAGEFRAMESIZE=MAX
PAGEFRAMESIZE= Pageable1MEG
PAGEFRAMESIZE= Pageable2MEG

An optional input parameter that specifies the size of the page frames to back the virtual storage mapped by the allocated memory object.

- PAGEFRAMESIZE=4K
  The memory object should be backed by 4 KB-page frames. The default value is PAGEFRAMESIZE=4K.

- PAGEFRAMESIZE=1MEG
  The memory object should be backed by 1 MB-page frames.

- PAGEFRAMESIZE=MAX
  The memory object should be backed by the largest page frame size supported but if the request cannot be backed by the largest frame size due to the availability of large page frames, then the request will be backed by 4 KB-page frames. 1 megabyte page frames are backed at allocation time and cannot be paged out to AUX. 4 KB-page frames are backed at first
The memory object is backed by pageable 1 MB-page frames at first reference, unless none are available. If none are available, the object is backed by 4 KB-page frames.

**PAGEFRAMESIZE=DREF1MEG**

The memory object is backed by pageable 1 MB-page frames at first reference, unless none are available. If none are available, the object is backed by 4 KB-page frames.

**TYPE=PAGEABLE**

An optional input parameter that specifies the type of storage that is requested. The default value is TYPE=PAGEABLE when one of the following parameters is specified:

- PAGEFRAMESIZE=PAGEABLE1MEG
- PAGEFRAMESIZE=4K
- PAGEFRAMESIZE=MAX and the memory object is backed with 4 KB-page frames.

The default value is TYPE=DREF when PAGEFRAMESIZE=DREF1MEG is specified. If PAGEFRAMESIZE=1MEG or PAGEFRAMESIZE=MAX is specified and the memory object is backed with 1 MB-page frames, the 1 MB-pages backing this memory object are fixed.

**Note:**

1. When the memory object is backed by 4 KB-page frames, the 4 KB-pages backing this memory object are pageable if TYPE=DREF is not specified; the 4 KB-pages are fixed if TYPE=DREF is specified. The 4 KB-pages are backed at first reference and can and can only be paged out to AUX if TYPE=DREF is not specified.

2. When the memory object is backed by 1 MB-page frames as a result of PAGEFRAMESIZE=PAGEABLE1MEG or PAGEFRAMESIZE=DREF1MEG being specified, the 1 MB-pages backing this memory object are pageable if PAGEABLE1MEG is specified or fixed if DREF1MEG is specified. Pageable 1 MB-pages are backed at first reference and can be paged out to AUX. DREF 1 MB-pages are backed at first reference and are fixed—they cannot be paged out to AUX.

3. When the memory object is backed by 1 MB-page frames because PAGEFRAMESIZE=1MEG or PAGEFRAMESIZE=MAX has been specified, the 1 MB-pages backing this memory object are fixed. Pages are backed at allocation time and cannot be paged out to AUX.

**TYPE=PAGEABLE**

Pages backing this memory object are pageable. Pages are backed at first reference and can be paged out to AUX. virtual address ranges within the memory object can be explicitly fixed after allocation by using the IARV64 REQUEST=PAGEFIX request.

**TYPE=DREF**

The memory object is referenced while running disabled. Note that the DREF attribute applies to the entire memory object. Pages are backed in real at first reference. Pages belonging to memory objects with the TYPE=DREF attribute remain in real and are never paged out to AUX.
**IARV64 macro**

,.UNITS=units

UNITS and SEGMENTS are mutually exclusive keys. This set is required; only one key can be specified.

A required input parameter that specifies the size of the memory object as a number of units specified by the UNITSIZE parameter. This must be a nonzero value. The amount of storage requested that is not in the guard state is counted towards the MEMLIMIT for the address space where the memory object will be created. UNITS belongs to a set of mutually exclusive keys. This set is required; only one key can be specified.

,.TYPE=PAGEABLE
,.TYPE=DREF
,.TYPE=FIXED

A required input parameter that specifies the type of requested storage.

,.TYPE=PAGEABLE
   Pages backing this memory object are pageable. Pages are backed at first reference and can be paged out to auxiliary storage. Virtual address ranges within the memory object can be explicitly fixed after allocation by using the IARV64 REQUEST=PAGEFIX request.
   TYPE=PAGEABLE is not valid with PAGEFRAMESIZE=2G.

,.TYPE=DREF
   Pages are backed in real memory at first reference, unless DREF storage is not available, in which case the program is ABENDed. Once backed, pages belonging to memory objects of TYPE=DREF remain in real storage and are never paged out to auxiliary storage. The memory object can be referenced while running disabled. The DREF attribute applies to the entire memory object. TYPE=DREF is not valid with PAGEFRAMESIZE=2G.

,.TYPE=FIXED
   Pages are backed in real storage immediately, unless fixed storage is not immediately available, in which case the request fails. Pages belonging to memory objects of TYPE=FIXED remain in real storage and are never be paged out to auxiliary storage. The memory object can be referenced while running disabled. The FIXED attribute applies to the entire memory object when it is allocated. TYPE=FIXED is not valid with PAGEFRAMESIZE=4K.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

,.UNITSIZE=1M
,.UNITSIZE=2G

A required input parameter that specifies the size for the UNITS parameter: either 1 MB or 2 GB.

,.UNITSIZE=1M
   Specifies that the memory object is in 1 MB units. For example, a request for UNITS=3 with UNITSIZE=1M is a request for three megabytes of virtual storage starting on a 1 MB boundary. When UNITSIZE=1M is specified, one of the following PAGEFRAMESIZE values must also be specified:
   PAGEFRAMESIZE=4K
PAGEFRAMESIZE=1M
A required input parameter that specifies the size of the page frames used to back the virtual storage mapped by the allocated memory object.

PAGEFRAMESIZE=4K
Specifies to back the memory object with 4 KB-page frames of the specified TYPE, when TYPE=PAGEABLE or TYPE=DREF is requested. TYPE=FIXED is not supported.

PAGEFRAMESIZE=1M
Specifies to back the memory object by one-megabyte (1 MB) page frames of the specified TYPE. If 1 MB-page frames are not supported or not available, the system attempts to back the memory object at a smaller page frame size of the specified TYPE, when TYPE=PAGEABLE or TYPE=DREF is requested. A TYPE=FIXED request fails if there are no available pages in the requested PAGEFRAMESIZE.

UNITSIZE=2G
Specifies that the memory object is in two-gigabyte (2G) units. For example, a request for UNITS=3 with UNITSIZE=2G is a request for six gigabytes of virtual storage starting on a 2 GB boundary.

PAGEFRAMESIZE=4K|1M
A required input parameter that specifies the size of the page frames that back the virtual storage mapped by the allocated memory object.

PAGEFRAMESIZE=4K
Specifies to back the memory object by 4 KB-page frames of the specified TYPE, when TYPE=PAGEABLE or TYPE=DREF is requested. TYPE=FIXED is not supported with this value.

PAGEFRAMESIZE=1M
Specifies to back the memory object by one-megabyte (1 MB) page frames of the specified TYPE. If 1 MB-page frames are not supported or not available when TYPE=PAGEABLE or TYPE=DREF is requested, the system attempts to back the memory object using a smaller page frame size of the specified TYPE. A TYPE=FIXED request fails if there are no available pages in the requested PAGEFRAMESIZE.

OWNERCOM=HOME
OWNERCOM=PRIMARY
OWNERCOM=SYSTEM
OWNERCOM=BYASID
An optional input parameter that specifies the entity to which the system will assign ownership of the 64-bit common memory object. The system uses this ownership information to track the use of 64-bit common storage for diagnostic purposes. The default is OWNERCOM=HOME.

OWNERCOM=HOME
The home address space will be assigned as the owner of the 64-bit common memory object.

OWNERCOM=PRIMARY
The primary address space will be assigned as the owner of the 64-bit common memory object.
The system (the 64-bit common memory object is not associated with an address space) will be assigned as the owner of the 64-bit memory object.

The address space specified by OWNERASID will be assigned as the owner of the 64-bit common memory object.

An optional input parameter that specifies the ASID of the address space that will own the 64-bit common memory object for tracking purposes. The default is OWNERASID=0.

This parameter indicates that the system is assigned as the owner of the 64-bit memory object.

This is the name (RS-Type), or address in register (2)-(12), of an optional halfword input that contains the address space identifier (ASID) to be designated as the owner of the 64-bit common memory object for storage tracking purposes.

The 64-bit common memory object is included in an SVC dump when CSA is specified on SDATA.

The 64-bit common memory object is included in an SVC dump when SQA is specified on SDATA.

The 64-bit common memory object is not included in an SVC dump when either CSA or SQA is specified on SDATA.

The 64-bit common memory object is dumped according to the option specified by the OPTIONVALUE keyword.

This parameter is the name (RS-Type), or address in register (2)-(12), of a required one-byte integer input that contains one of the following:

- XMFCTRL_XDUMP_NO - (X’01’) – this is equivalent to DUMP=NO
- XMFCTRL_XDUMP_LIKESQA - (X’02’) – this is equivalent to DUMP=LIKESQA
- XMFCTRL_XDUMP_LIKECSA - (X’03’) – this is equivalent to DUMP=LIKECSA

The 64-bit common memory object is not included in an SVC dump when either CSA or SQA is specified on SDATA.
IARV64 macro

,DETACHFIXED=YES
An optional input parameter that specifies whether the memory object can be
detached when it contains fixed pages at the time of the DETACH request. The
default value for DETACHFIXED is NO.

DETACHFIXED=NO
The memory object will not be detached if it has any fixed pages when it is
detached.

,ORIGIN=origin
A required output parameter that contains the lowest address of the memory
object.

Note: When GUARDLOC=LOW is specified, the lowest address will point to a
guard area which will cause an ABEND if referenced. For GUARDLOC=LOW
the first usable area is the origin plus the size of the guard area.

To code: Specify the RS-type address or address in register (2) - (12) of an
eight-byte pointer field.

,RETCODE=retcode
An optional output parameter into which the return code is to be copied from
GPR 15.

To code: Specify the RS-type address of a fullword field or register (2) - (12).

,RSNCODE=rsncode
An optional output parameter into which the reason code is to be copied from
GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4, 5
An optional input parameter that specifies the version of the macro. PLISTVER
determines which parameter list the system generates. PLISTVER is an
optional input parameter on all forms of the macro, including the list form.
When using PLISTVER, specify it on all macro forms used for a request and
with the same value on all of the macro forms. The values are:

• IMPLIED_VERSION, which is the lowest version that allows all parameters
  specified on the request to be processed. If you omit the PLISTVER
  parameter, IMPLIED_VERSION is the default

• MAX, if you want the parameter list to be the largest size currently possible.
  This size might grow from release to release and affect the amount of
  storage that your program needs.

If you can tolerate the size change, IBM suggests that you always specify
PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that
the list-form parameter list is always long enough to hold all the parameters
you might specify on the execute form, when both are assembled with the
same level of the system. In this way, MAX ensures that the parameter list
does not overwrite nearby storage.

• 0, supports all parameters except those specifically referenced in higher
  versions.

• 1, supports both the following parameters and parameters from version 0:
IARV64 macro

- CONVERTSIZE64
- CONVERTSTART
- GUARDSIZE64
- GETSHARED

- 2, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON

- 3, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO

- 4, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPagetable

- 5, supports both the following parameters and parameters from versions 0, 1, 2, 3, 4:
  - UNITS
  - UNITSIZE

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4 or 5

, MF=\$S
, MF=(L, list addr)
, MF=(L, list addr, attr)
, MF=(L, list addr, BD)
, MF=(E, list addr)
, MF=(E, list addr, COMPLETE)
, MF=(M, list addr)
, MF=(M, list addr, COMPLETE)
IARV64 macro

\( MF=(M, list \ addr, NOCHECK) \)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:
- Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,list_addr,NOCHECK), to execute the macro.

\( list \ addr \)

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

\( attr \)

An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

\( COMPLETE \)

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

\( NOCHECK \)

This parameter specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

REQUEST=COUNTPAGES option of IARV64

REQUEST=COUNTPAGES requests the count of the number of 4K pages currently in use in real storage, on auxiliary storage, and in both real storage and on auxiliary storage to back the input high virtual storage ranges.
IARV64 macro

**Environment**

The requirements for the caller are:

- **Environmental factor**
- **Minimum authorization:** Supervisor state or with PSW key 0-7.
  - The caller must be running in supervisor state or with PSW key 0-7.
- **Dispatchable unit mode:** Task or SRB
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 31- or 64-bit
- **ASC mode:** Primary or access register (AR)
- **Interrupt status:** Enabled for I/O and external interrupts.
- **Locks:** A local lock may be held.
- **Control parameters:** Control parameters must be in the primary address space and can reside both below and above the bar.

**Programming requirements**

None

**Restrictions**

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

**Input register information**

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

**Output register information**

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if GPR 15 is non-zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None

**Syntax**

The REQUEST=COUNTPAGES option of the IARV64 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede IARV64.</td>
</tr>
<tr>
<td>IARV64</td>
<td></td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must follow IARV64.</td>
</tr>
<tr>
<td>REQUEST=COUNTPAGES</td>
<td></td>
</tr>
<tr>
<td>,COND=NO</td>
<td>Default: COND=NO</td>
</tr>
<tr>
<td>,COND=YES</td>
<td></td>
</tr>
<tr>
<td>,V64LISTPTR=v64listptr</td>
<td><code>v64listptr</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,V64LISTLENGTH=v64listlength</td>
<td><code>v64listlength</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,RANGLIST=ranglist</td>
<td><code>ranglist</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,NUMRANGE=numrange</td>
<td><code>numrange</code>: RS-type address or address in register (2) - (12). Default: NUMRANGE=1</td>
</tr>
<tr>
<td>,ALETVALUE=aletvalue</td>
<td><code>aletvalue</code>: RS-type address or address in register (2) - (12). Default: ALETVALUE=0</td>
</tr>
<tr>
<td>,UNLOCKED=NO</td>
<td>Default: NO</td>
</tr>
<tr>
<td>,UNLOCKED=YES</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td><code>retcode</code>: RS-type address or register (2) - (12) or (15), (GPR15).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td><code>rsncode</code>: RS-type address or register (0) or (2) - (12).</td>
</tr>
</tbody>
</table>
IARV64 macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>.PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>.PLISTVER=0, 1, 2, 3</td>
<td></td>
</tr>
<tr>
<td>.MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>.MF=(L, list addr)</td>
<td>list addr: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td>.MF=(L, list addr, attr)</td>
<td></td>
</tr>
<tr>
<td>.MF=(L, list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>.MF=(E, list addr)</td>
<td></td>
</tr>
<tr>
<td>.MF=(E, list addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

name

An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=COUNTPAGES

A required parameter. REQUEST=COUNTPAGES requests the count of the number of 4K pages currently in use in real storage, on auxiliary storage, and in both real storage and on auxiliary storage to back the input high virtual storage ranges. The counts are returned as grand totals for all pages within the ranges requested. If a page is in real storage and on auxiliary storage, it will be counted in real storage, on auxiliary storage, and in both real storage and on auxiliary storage. See IAXV64WA for a description of the output area.

,COND=NO

An optional parameter that specifies whether the request is unconditional or conditional. In all cases, the request will be abnormally ended for invalid requests, including violation of environmental restrictions. The default is COND=NO.

,COND=YES

The request is conditional. The request is not abnormally ended for resource unavailability.

,V64LISTPTR=v64listptr

A required input parameter that contains the address of the work area that will contain the results of the COUNTPAGES request. The work area must be in fixed storage addressable from the address space for which the request is made and must be initialized to zero by the caller.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.
V64LISTLENGTH=v64listlength
A required input parameter that specifies the length of the work area that contains the results of the COUNTPAGES request. The work area must be at least 64 bytes long.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

RANGLIST=ranglist
A required input parameter that contains the address of a range list. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry is as follows:

VSA
Denotes the starting virtual address of the data to be acted on. The virtual address must be on a page boundary and within a memory object returned by GETSHARED, GETSTOR or GETCOMMON.

The length of this field is 8 bytes.

NUMPAGES
Contains the number of 4K pages to be acted on. The number of pages specified, starting with the specified VSA, must lie within a single memory object.

The length of this field is 8 bytes.

To code: Specify the RS-type address, or address in register (2)-(12), of a 8-byte pointer field.

NUMRANGE=numrange
NUMRANGE=1
An optional input parameter that specifies the number of entries in the supplied range list up to 16. The default is NUMRANGE=1.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

ALETVALUE=aletvalue
ALETVALUE=0
An optional input parameter that indicates the ALET of the address space that will access the memory object.

The only supported values are 0 (primary) and 2 (home).

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

UNLOCKED=NO
UNLOCKED=YES
An optional parameter that specifies whether the request is to be performed without RSM serialization. This should only be used when the memory objects to be counted will not be modified or detached and an exact count is not needed. These counts will not be accurate because RSM processing can be modifying frame allocations. The default is UNLOCKED=NO.

- UNLOCKED=NO: The request will use RSM serialization. This is the recommended option.
- UNLOCKED=YES: The request will not hold RSM locks during processing.

RETCODE=retcode
An optional output parameter into which the return code is to be copied from
IARV64 macro

GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12), (15), (GPR15), (REG15), or (R15)

,RSNCD=rsncode
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (0), (GPR0), (GPR00), (REG0), (REG00), or (R0).

,PLI$TVER=IMPLIED_VERSION
,PLI$TVER=MAX
,PLI$TVER=0, 1, 2, 3
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

• IMPLIED_VERSION, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
• MAX, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs. If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
• 0, supports all parameters except those specifically referenced in higher versions.
• 1, supports both the following parameters and parameters from version 0:
  − CONVERSIONSIZE64
  − CONVERSIONSTART
  − GUARDSIZE64
  − V64SHARED

• 2, supports both the following parameters and parameters from version 0 and 1:
  − AMOUNTSIZE
  − DETACHFIXED
  − DUMP
  − DUMPPRIORITY
  − DUMPPROTocol
  − LOCALSYSAREA
  − MEMLIMIT
  − OPTIONVALUE
  − ORDER
  − OWNERASID
IARV64 macro

- OWNERCOM
- TYPE
- USERTOKEN
- V64COMMON

- 3, supports both the following parameters and parameters from version 0, 1 and 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2 or 3.

,MF=S
,MF=(L,list addr)
,MF=(L,list addr,attr)
,MF=(L,list addr,8D)
,MF=(E,list addr)
,MF=(E,list addr,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

IARV64 might abnormally terminate with hexadecimal abend code DC2. See DC2 in z/OS MVS System Codes for an explanation and programmer response.
Return and reason codes

When the IARV64 macro returns control to your program GPR 15 (and retcode, when you code RETCODE) contains a return code. When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes. IBM support personnel may request the entire reason code, including the xx value.

Table 37. Return and Reason Codes for the IARV64 Macro

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>—</td>
<td>Meaning: Successful completion Action: None required</td>
</tr>
<tr>
<td>02</td>
<td>—</td>
<td>Meaning: Successful completion, with exception. For a LIST request, IARV64 requests have been issued since the previous call to LIST. Action: Reissue the call if you need the information pertaining to those recent IARV64 requests.</td>
</tr>
<tr>
<td>04</td>
<td>—</td>
<td>Meaning: Successful completion, with exception. For a LIST request, there are additional memory objects which were not returned on this call to LIST. For a CHANGEGUARD request, one or more segments in the memory object are already in the requested state. For a DETACH request, the memory object task token does not match the TToken of the caller. For GETSTOR, GETCOMMON request with PAGEFRAMESIZE(MAX), no large frame was available, PAGEFRAMESIZE(4K) was used. Action: For a CHANGEGUARD request, if this is unexpected, then ensure that the correct values for CONVERTSIZE, CONVERTSIZE64, or CONVERTSTART are specified. If it is already known that some segments may be in the requested state, then no action is required. For a LIST request, issue the LIST call again to get the additional information. For GETSTOR, GETCOMMON request with PAGEFRAMESIZE(MAX), no action is required, the memory object was backed by a 4K page.</td>
</tr>
</tbody>
</table>
### Table 37. Return and Reason Codes for the IARV64 Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>06</td>
<td>—</td>
<td><strong>Meaning:</strong> Successful completion, with exception. For a LIST request, there are additional memory objects which were not returned on this call to LIST and IARV64 requests have been issued since the previous call to LIST. For a COUNTPAGES request, the counts are incomplete because additional IARV64 requests have been issued during COUNTPAGES processing. <strong>Action:</strong> Issue the LIST or COUNTPAGES call again to get the additional information.</td>
</tr>
<tr>
<td>08</td>
<td>—</td>
<td><strong>Meaning:</strong> The request is rejected because of non-system failure. This reason code could be issued for a conditional IARV64 request. In this case this reason code is the same as the DC2 reason code issued from an unconditional IARV64 request. See DC2 in <a href="http://www.ibm.com">z/OS MVS System Codes</a> for an explanation and programmer response. Otherwise, if it is not there, then it has one of the following meanings: For a DETACH request, there were no memory objects deleted because none matched the user token provided. For a LIST request, there were no memory objects returned because no memory objects match the selection criteria. <strong>Action:</strong> For a DETACH request, make sure that the user token was correct. For a LIST request, no action is required. For other requests, see DC2 in <a href="http://www.ibm.com">z/OS MVS System Codes</a> for an explanation and programmer response.</td>
</tr>
</tbody>
</table>
### Table 37. Return and Reason Codes for the IARV64 Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0C          | —           | **Meaning:** The request is rejected because of system failure. 

This reason code could be issued for a conditional IARV64 request. In this case this reason code is the same as the DC2 reason code issued from an unconditional IARV64 request. See DC2 in [z/OS MVS System Codes](#) for an explanation and programmer response. Otherwise, if it is not there, then it has the following meaning:

- For a GETSTOR request, there was insufficient storage to build the control structure.
- For a COUNTPAGES request, there was an error during UNLOCKED=YES processing that indicates the page table structure has changed.

**Action:**

- For a GETSTOR request, free storage within address space so control structures can be built.
- For a COUNTPAGES request, try the request again.
- For all other requests, see DC2 in [z/OS MVS System Codes](#) for an explanation and programmer response.

### Example

**Operation:**

1. Get 2 MB above the bar
2. Page-fix the first 1 MB of that storage
3. Page-unfix that first 1 MB
4. Free the storage

The code is as follows:

```assembly
SYSSTATE AMODE64=YES
************************************************************
* Get storage above 2G *
************************************************************
IARV64 REQUEST=GETSTOR,SEGMENTS=NUMSEG, *
ORIGIN=OUTORG,RETCODE=LRETCODE, *
RSNCODE=LRSNCODE,CONTROL=AUTH, *
MF=(E,V64L) *
* Place code to check return/reason codes here *
* *
* Build the Range List for Pagefix *
* 
LG 1,OUTORG 
STG 1,RLSTART 
LG 1,ONEMEG Number of pages in 1-meg 
STG 1,RLEND 
LA 1,RL 
LLGTR 1,1
```

---

This response is based on the extracted text content provided. It includes the detailed table of return and reason codes for the IARV64 macro, along with an example operation and the associated assembly code. The table entries are explained in detail, and the example follows the specified operation steps, providing a clear demonstration of how to use the IARV64 macro in practice.
IARV64 macro

STG 1,RLADDR

**********************************************************
* Page-fix that storage                                  *
* Defaults to NUMRANGE=1.                                *
**********************************************************

* IARV64 REQUEST=PAGEFIX,RANGLIST=RLADDR,                *
  RETCODE=LRETCODE,RSNCODE=LRSNCODE,                   *
  MF=(E,V64L)                                          *
*
* Place code to check return/reason codes here           *
*
**********************************************************
* Page-unfix that storage                               *
* Defaults to NUMRANGE=1.                                *
**********************************************************

* IARV64 REQUEST=PAGEUNFIX,RANGLIST=RLADDR,              *
  RETCODE=LRETCODE,RSNCODE=LRSNCODE,                   *
  MF=(E,V64L)                                          *
*
* Place code to check return/reason codes here           *
*
**********************************************************
* Free the storage                                      *
**********************************************************

* IARV64 REQUEST=DETACH,MEMOBJSTART=OUTORG              *
  RETCODE=LRETCODE,RSNCODE=LRSNCODE,                   *
  MF=(L,V64L)                                          *
*
* Place code to check return/reason codes here           *
*
NUMSEG DC AD(2)
ONEMEG DC AD((1024*1024)/4096) Num of pages in a megabyte
DYNAREA DSECT
LRETCODE DS F
LRSNCODE DS F
OUTORG DS AD
RLADDR DS OD Start of 16-byte range list
RLSTART DS AD Address of memory object
RLEND DS AD Number of pages
IARV64 MF=(L,V64L)
IARV64 macro
Chapter 30. IAZXCTKN — Client token compare service

Description

Use the IAZXCTKN macro to compare two client tokens. This service should be used anytime client tokens have to be compared.

Environment

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state, any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td></td>
<td>Secondary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller may hold locks, but is not required to hold any.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>None</td>
</tr>
</tbody>
</table>

Programming requirements

Include the IAZXCTKN mapping macro.

Restrictions

None.

Input register information

Before issuing the IAZXCTKN macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Destroyed</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Destroyed</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

Performance implications

None.
IAZXCTKN macro

Syntax

The IAZXCTKN macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTOKENA=ctoken addr</td>
<td>ctoken addr: RX-type address or address in register (2) - (13).</td>
</tr>
<tr>
<td>CTOKENB=ctoken addr</td>
<td>ctoken addr: RX-type address or address in register (2) - (13).</td>
</tr>
</tbody>
</table>

Parameters

The IAZXCTKN parameters are explained as follows:

- **CTOKENA=ctoken addr**
  - Specifies the address of an 80-byte area containing one CTOKEN to be compared.

- **CTOKENB=ctoken addr**
  - Specifies the address of an 80-byte area containing one CTOKEN to which CTOKENA is to be compared.

ABEND codes

None.

Return codes

When IAZXCTKN macro returns control to your program, GPR 15 contains a return code.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: CTOKENA and CTOKENB contain the same significant information.</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: CTOKENA and CTOKENB contain different significant information and CTOKENA's sort information is less than CTOKENB's sort information.</td>
</tr>
<tr>
<td>08</td>
<td>Meaning: CTOKENA and CTOKENB contain different significant information and CTOKENB's sort information is less than CTOKENA's sort information.</td>
</tr>
<tr>
<td>12</td>
<td>Meaning: CTOKENA and CTOKENB contain different significant information and at least one of these ctokens contains no sort information.</td>
</tr>
<tr>
<td>16</td>
<td>Meaning: CTOKENA and CTOKENB contain different significant information but the sort information in the two ctokens is equal. This indicates a &quot;collision&quot; of the two ctokens.</td>
</tr>
</tbody>
</table>

Example

IAZXCTKN CTOKENA=BILL,CTOKENB=SAM
Chapter 31. IAZXJSAB — Obtain information about a currently running job

Description

Use the CREATE function with TYPE=SUBTASK to set or request job scheduler information for the current task. When an application is doing work on behalf of another work unit, the CREATE or UPDATE request identifies the work unit for whom the work is being done. IAZXJSAB provides the following services:

- Create a JSAB.
- Read from a JSAB.
- Update a JSAB.
- Delete a JSAB.

Environment

The requirements for the caller are:

Environmental factor: Requirement
Minimum authorization: CREATE, DELETE and UPDATE: APF authorization, supervisor state or system key
READ: Problem or supervisor state and any PSW key
Dispatchable unit mode: Task or SRB
Cross memory mode: If the caller specifies the ASCB parameter, any PASN, any HASN, any SASN; otherwise, PASN=HASN is required.
AMODE: 31-bit
ASC mode: Primary or access register (AR)
Interrupt status: Enabled or disabled for I/O and external interrupts
Locks: The caller may hold locks, but is not required to hold any.
Control parameters: None

Programming requirements

Include these mapping macros in your program: IAZJSAB, IHAASCB and IHAASSB. If you do not code the ASCB parameter, also include these additional mapping macros: IHAPSA, IKJTCB, and IHASTCB.

If you need more information about:

<table>
<thead>
<tr>
<th>Mapping macro</th>
<th>Look in:</th>
<th>Under the name:</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAZJSAB</td>
<td>z/OS MVS Data Areas in the <a href="http://www.ibm.com/systems/z/os/zos/bkserv/">z/OS Internet library</a></td>
<td>JSAB</td>
</tr>
<tr>
<td>IHAASCB</td>
<td>z/OS MVS Data Areas in the <a href="http://www.ibm.com/systems/z/os/zos/bkserv/">z/OS Internet library</a></td>
<td>ASCB</td>
</tr>
<tr>
<td>IHAASSB</td>
<td>z/OS MVS Data Areas in the <a href="http://www.ibm.com/systems/z/os/zos/bkserv/">z/OS Internet library</a></td>
<td>ASSB</td>
</tr>
<tr>
<td>IHAPSA</td>
<td>z/OS MVS Data Areas in the <a href="http://www.ibm.com/systems/z/os/zos/bkserv/">z/OS Internet library</a></td>
<td>PSA</td>
</tr>
<tr>
<td>IHASTCB</td>
<td>z/OS MVS Data Areas in the <a href="http://www.ibm.com/systems/z/os/zos/bkserv/">z/OS Internet library</a></td>
<td>STCB</td>
</tr>
<tr>
<td>IKJTCB</td>
<td>z/OS MVS Data Areas in the <a href="http://www.ibm.com/systems/z/os/zos/bkserv/">z/OS Internet library</a></td>
<td>TCB</td>
</tr>
</tbody>
</table>
IAZXJSAB macro

Restrictions
The following restrictions apply:
• You must not create an address space JSAB.
• You may only update or delete a JSAB that you have created.

Input register information
Before issuing the IAZXJSAB macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
The IAZXJSAB macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IAZXJSAB macro.</td>
</tr>
<tr>
<td>IAZXJSAB</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IAZXJSAB macro.</td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td></td>
</tr>
<tr>
<td>UPDATE</td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### ,TYPE=SUBTASK

**Default:** ADDRSP (Restricted use)

#### ,ASCB=ascb addr

**ascb addr:** RX-type address or register (2) - (12).

**Default:** The address of the ASCB for the caller's home address space.

#### ,COMPID=compid addr

**compid addr:** RS-type address or register (2) - (12).

#### ,CODELEV=codelev addr

**codelev addr:** RX-type address or register (2) - (12).

#### ,WORKID=workid addr

**workid addr:** RS-type address or register (2) - (12).

#### ,JOBID=jobid addr

**jobid addr:** RS-type address or register (2) - (12).

#### ,JOBNAME=jobname addr

**jobname addr:** RS-type address or register (2) - (12).

#### ,PREFIX=prefix addr

**prefix addr:** RS-type address or register (2) - (12).

#### ,USERID=userid addr

**userid addr:** RS-type address or register (2) - (12).

#### ,EXECST=execst addr

**execst addr:** RS-type address or register (2) - (12).

#### ,XCFGPNM=xcfgpnm addr

**xcfgpnm addr:** RS-type address or register (2) - (12).

#### ,JESTAT=jestat addr

**jestat addr:** RS-type address or register (2) - (12).

#### ,JSABLVL=jsablvl addr

**jsablvl addr:** RS-type address or register (2) - (12).

#### ,JOBCORR=jobcorr addr

**jobcorr addr:** RS-type address or register (2) - (12).

---

## Parameters

The parameters are explained as follows:

**CREATE | READ | UPDATE | DELETE**

Requests the type of IAZXJSAB function.
IAZXJSAB macro

,TYPE=SUBTASK
   Specifies the type of JSAB to be created or deleted. TYPE is valid only for the
   CREATE and DELETE services. You must code TYPE=SUBTASK, as the default
   is TYPE=ADDRSP.

,ASCB=ascb addr
   Specifies the address of an address space control block (ASCB).
   ASCB is valid only for the READ and DELETE services. The default value is
   the address of the ASCB that represents the caller's home address space.

,COMPID=compid addr
   With the CREATE service, COMPID specifies the location of the 4-character
   name of the subsystem that is creating the JSAB. With the READ service,
   COMPID specifies the location where the system is to return the 4-character
   name of the subsystem that created the JSAB. If JES2 or JES3 created the JSAB,
   the identifier is the common name of the JES (such as JES2 or JES3) and not
   the name of the JES2 or JES3 address space. If APPC/MVS created the JSAB,
   the identifier is ASCH.
   COMPID is required for the CREATE service and is optional for the READ
   service. COMPID is not valid on the UPDATE and DELETE services.

,CODELEV=codelev addr
   Specifies the code level of the creating component. Valid values are 0-255.
   CODELEV is required for the CREATE service and is optional for the READ
   service. CODELEV is not valid on the UPDATE and DELETE services.

,WORKID=workid addr
   Specifies the location where the system is to return the 8-character work unit
   identifier. The system returns identical information for the work unit ID and
   job ID.
   WORKID is not valid on the DELETE service and is optional on the CREATE,
   READ, and UPDATE services.

,JOBNAMExjobname addr
   Specifies the location where the system is to return the 8-character job
   identifier. The system returns identical information for the work unit ID and
   job ID.
   JOBNAMEx is not valid on the DELETE service and is optional on the CREATE,
   READ, and UPDATE services.

,PREFIX=prefix addr
   Specifies the location where the system is to return the 8-character message
   prefix. In a JES2 system, the prefix is the job ID. In a JES3 system, the prefix is
   the job name.
   PREFIX is not valid on the DELETE service and is optional on the CREATE,
   READ, and UPDATE services.

,USERID=userid addr
   Specifies the location where the system is to return the 8-character user ID.
   USERID is not valid on the DELETE service and is optional on the CREATE,
   READ, and UPDATE services.
IAZXJSAB macro

,EXECS=execst addr
Specifies the location where the system is to return the 8-byte execution start time, in time-of-day (TOD) clock format.

EXECS is not valid on the DELETE service.

,XCFGPNM=xcfgpnm addr
Specifies the location where the system is to return the 8-character XCF group name of the subsystem that created the JSAB. The XCF group name is available only if JES2 created the JSAB.

XCFGPNM is not valid on the DELETE service and is optional on the CREATE, READ, and UPDATE services.

,JESTAT=jestat addr
Specifies the location where the system is to return the 8-byte JES status for the address space.

For the meaning of values that can be returned to the specified address, see the field JSABJSTA in the mapping macro IAZJSAB in z/OS MVS Data Areas in the z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).

JES status only applies on the job level. Therefore, using the JESTAT keyword forces the access to the address space level JSAB. JESTAT is not compatible with TYPE(SUBTASK). For READ and UPDATE, JESTAT can only be used with the JOBCORR keyword—JESTAT cannot be used together with any other job attribute.

,JSABLVL=jsablvl addr
Specifies the location where the system is to return the 1-byte level of the JSAB to be used for the READ request. If the value returned is 4, it indicates a subtask level JSAB.

JSABLVL is valid only on the READ service.

,JOBCORR=jobcorr addr
Specifies the location where the system is to return the 64-character job correlator field. Because JOBCORR only applies on the job level, using the JOBCORR keyword forces the access to the address space level JSAB.

JOBCORR is not compatible with TYPE(SUBTASK). For READ and UPDATE, JOBCORR can only be used with the JESTAT keyword—JOBCORR cannot be used with any other job attribute.

JOBCORR is optional for CREATE, READ and UPDATE. JOBCORR is not valid for DELETE.

ABEND codes
None.

Return codes
When IAZXJSAB macro returns control to your program, GPR 15 contains one of the following hexadecimal return codes.

Table 39. Return and Reason Codes for the IAZXJSAB Macro

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Meaning: Processing completed successfully.</td>
</tr>
<tr>
<td></td>
<td>Action: None.</td>
</tr>
</tbody>
</table>
### IAZXJSAB Macro

#### Table 39. Return and Reason Codes for the IAZXJSAB Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Storage was not obtained or released for the JSAB. (CREATE and DELETE only.)</td>
<td>None.</td>
</tr>
<tr>
<td>8</td>
<td>The JSAB was not found. No information was returned.</td>
<td>None required; however, you might want to make sure the specified ASCB address is correct.</td>
</tr>
<tr>
<td>12</td>
<td>The requested field does not exist in the active JSAB.</td>
<td>None.</td>
</tr>
</tbody>
</table>

#### Example

Obtain the job ID of the current address space.

IAZXJSAB READ, JOBID=MYJOBID

```
.
.
.
MYJOBID  DS  CL8
```
Chapter 32. IEAARR — Establish an associated recovery routine (ARR)

Description

IEAARR allows you to request that the system establish an associated recovery routine (ARR) while calling a target routine. In this case, the system performs the stacking PC instruction, then give control to your routine (the target routine). When the target routine returns control, the system issues the corresponding PR instruction.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and PSW key 8-15</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller is not required to hold any locks on entry. The caller may hold the local, CMS, or CPU lock.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>None.</td>
</tr>
</tbody>
</table>

Programming requirements

The caller must include the IHAECVT mapping macro.

Restrictions

IEAARR must not be issued while a functional recovery routine (FRR) is established.

TARGETSTATE=PROB should only be issued by a caller currently running in problem state. TARGETSTATE=SUP should only be issued by a caller currently running in supervisor state.

Input register information

Before issuing the IEAARR macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The value placed in register 0 by the target routine prior to its returning to the system.</td>
</tr>
</tbody>
</table>
IEAARR macro

1 The value placed in register 1 by the target routine prior to its returning to the system.
2-13 Unchanged
14 Used as a work register by the system
15 The value placed in register 15 by the target routine prior to its returning to the system.

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The value placed in access register 0 by the target routine prior to its returning to the system.</td>
</tr>
<tr>
<td>1</td>
<td>The value placed in access register 1 by the target routine prior to its returning to the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>The value placed in access register 15 by the target routine prior to its returning to the system.</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**Syntax**

The IEAARR macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td><strong>b</strong></td>
<td>One or more blanks must precede IEAARR.</td>
</tr>
<tr>
<td><strong>IEAARR</strong></td>
<td></td>
</tr>
<tr>
<td><strong>b</strong></td>
<td>One or more blanks must follow IEAARR.</td>
</tr>
<tr>
<td><strong>ARRPTR=arrptr</strong></td>
<td><em>arrptr</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><strong>ARR=arr</strong></td>
<td><em>arr</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><strong>,DYNSTORAGE=AVAIL</strong></td>
<td>Default: DYNSTORAGE=AVAIL</td>
</tr>
<tr>
<td><strong>,DYNSTORAGE=NOTAVAIL</strong></td>
<td></td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEAARR macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

DYNSTORAGE=AVAIL
DYNSTORAGE=NOTAVAIL
An optional parameter that indicates whether this routine is sensitive to your having dynamic storage. The default is DYNSTORAGE=AVAIL.

DYNSTORAGE=AVAIL
Indicates that you have dynamic storage available.

DYNSTORAGE=NOTAVAIL
Indicates that you do not have dynamic storage available. The parameters are defined so that you can place each parameter value in a register and in so doing avoid the need to place parameter values into dynamic storage.

ARRPTR=arrptr
When DYNSTORAGE=AVAIL is in effect, a required input parameter that contains the address of the associated recovery routine. This routine gets control from RTM according to normal z/OS recovery protocols. As it is an ARR, it will get control in AMODE 31.

To code: Specify the RX-type address, or address in register (2)-(12), of a pointer field.
IEAARR macro

\ARR=arr
When DYNSTORAGE=NOTAVAIL is specified, a required input parameter that is the associated recovery routine. This routine gets control from RTM according to normal z/OS recovery protocols. As it is an ARR, it will get control in AMODE 31.

**To code:** Specify the RX-type address, or address in register (2)-(12), of the associated recovery routine.

\ARRPARAMPTR=arrparamptr
When DYNSTORAGE=AVAIL is in effect and SYSSTATE AMODE64=YES is not in effect, a required input parameter that contains the address of the parameter area that is to be passed to the ARR upon error. The address is placed in the first four bytes of the area pointed to by SDWAPARM and in GPR 2. Note that the second four bytes of the area pointed to by SDWAPARM will not contain interface information.

**To code:** Specify the RX-type address, or address in register (2)-(12), of a pointer field.

\ARRPARAM=arrparam
When DYNSTORAGE=NOTAVAIL is specified and SYSSTATE AMODE64=YES is not in effect, a required input parameter that is the parameter area that is to be passed to the ARR upon error. The address is placed in the first four bytes of the area pointed to by SDWAPARM and in GPR 2. Note that the second four bytes of the area pointed to by SDWAPARM will not contain interface information.

**To code:** Specify the RX-type address, or address in register (2)-(12), of the parameter area.

\ARRPARAMPTR64=arrparamptr64
When DYNSTORAGE=AVAIL is in effect and SYSSTATE AMODE64=YES is in effect, a required 8-byte input parameter that contains the address of the parameter area that is to be passed to the ARR upon error. The address is placed in the 8-byte area pointed by SDWAPARM and in the 64-bit GPR 2.

**To code:** Specify the RX-type address, or address in register (2)-(12), of a 64-bit pointer field.

\ARRPARAM64=arrparam64
When DYNSTORAGE=NOTAVAIL is specified and SYSSTATE AMODE64=YES is in effect, a required 8-byte input parameter that is the parameter area that is to be passed to the ARR upon error. The address is placed in the 8-byte area pointed by SDWAPARM and in the 64-bit GPR 2.

**To code:** Specify the RX-type address, or address in register (2)-(12), of the parameter area.

\PARAMPTR=paramptr
When DYNSTORAGE=AVAIL is in effect and SYSSTATE AMODE64=YES is not in effect, a required input parameter that contains the address of a parameter that is to be passed to the target routine in GPR 1.

**To code:** Specify the RX-type address, or address in register (2)-(12), of a pointer field.

\PARAM=param
When DYNSTORAGE=NOTAVAIL is specified and SYSSTATE AMODE64=YES is not in effect, a required input parameter that is the parameter that is to be passed to the target routine in GPR 1.
**IEAARR macro**

To code: Specify the RX-type address, or address in register (2)-(12), of the parameter.

,PARAMPTR64=paramptr64

When DYNSTORAGE=AVAIL is in effect and SYSSTATE AMODE64=YES is in effect, a required 8-byte input parameter that contains the address of the parameter that is to be passed to the target routine in 64-bit GPR 1.

To code: Specify the RX-type address, or address in register (2)-(12), of a 64-bit pointer field.

,PARAM64=param64

When DYNSTORAGE=NOTAVAIL is specified and SYSSTATE AMODE64=YES is in effect, a required 8-byte input parameter that is to be passed to the target routine in 64-bit GPR 1.

To code: Specify the RX-type address, or address in register (2)-(12), of the parameter.

,TARGETPTR=targetptr

When DYNSTORAGE=AVAIL is in effect, a required input parameter that contains the address of the routine to which the system is to branch after establishing the ARR. The target routine will get control in the same key and state as the IEAARR caller, in AMODE 31, with the following input registers:

General Purpose Registers:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not part of the intended interface</td>
</tr>
<tr>
<td>1</td>
<td>Address of parameter area provided by IEAARR caller</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged from the IEAARR caller</td>
</tr>
<tr>
<td>14</td>
<td>The return address</td>
</tr>
<tr>
<td>15</td>
<td>The address of the target routine</td>
</tr>
</tbody>
</table>

Access Registers:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Not part of the intended interface</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged from the IEAARR caller</td>
</tr>
<tr>
<td>14</td>
<td>Not part of the intended interface</td>
</tr>
<tr>
<td>15</td>
<td>Not part of the intended interface</td>
</tr>
</tbody>
</table>

The target routine gets control with one more entry on the linkage stack than existed when IEAARR was called. That linkage stack entry contains the caller’s registers 2-13 which can be extracted using the EREG instruction if needed.

The target routine need not save any registers, but is expected to return to the address provided in GPR 14 on entry. The target routine can pass information back to the caller of IEAARR by placing it in GPR/AR 0, 1, and/or 15. The IEAARR caller will resume immediately after the IEAARR macro expansion.

To code: Specify the RX-type address, or address in register (2)-(12), of a pointer field.

,TARGET=target

When DYNSTORAGE=NOTAVAIL is specified, a required input parameter that
IEAARR macro

is the routine to which the system is to branch after establishing the ARR. The target routine interface is identical to that described under the TARGETPTR parameter.

**To code:** Specify the RX-type address, or address in register (2)-(12), of the target routine.

, TARGETSTATE=PROB, TARGETSTATE=SUP

A required parameter that indicates the requested PSW state of the target routine.

, TARGETSTATE=PROB
  indicates the target routine is to get control in problem state. This should only be used by a caller currently in problem state.

, TARGETSTATE=SUP
  indicates the target routine is to get control in supervisor state. This should only be used by a caller currently in supervisor state.

**ABEND codes**

The caller may get the following abend code:

0C2-02
  TARGETSTATE=SUP was requested by a caller currently running in problem state.

**Return codes**

None.

**Example 1**

**Operation:**

Branch to the target routine pointed to by field TP, and establish as an ARR the routine pointed to by field AP. Pass to the target area in register 1 the contents of field PP. Make sure that the ARR will get access to the contents of field APP (which ordinarily would contain the address of a parameter area). Make sure that the target routine gets control in problem state (which implies that the caller of IEAARR should currently be running in problem state).

The code is as follows.

```assembly
IEAARR TARGETPTR=TP,ARRPTR=AP,PARAMPTR=PP,
  ARRPARAMPTR=APP,TARGETSTATE=PROB
  ...
```
Chapter 33. IEAFP — Floating point services

Description

IEAFP allows you to request that, for your work unit, the system will stop saving additional floating point status. This status consists of the additional floating point registers (FPRs) 1, 3, 5, 7-15 and the floating point control (FPC) register. In addition, the system will stop saving vector register status.

You would typically use this service only when you are a server task which “subdispatches” unrelated units of work (that is, CICS transactions). To avoid subsequent units of work being penalized by the floating point actions of previous units of work, the additional FP status saving function of the operating system can be turned off. When a unit of work actually begins to use FP, all appropriate status saving will be resumed.

IEAFP allows you to request that the system stop saving vector register status, while continuing to save additional floating point status.

IEAFP allows you to request that the system start saving vector register status.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and PSW key 8-15</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller is not required to hold any locks on entry. The caller may hold the local, CMS, or CPU lock.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>None</td>
</tr>
</tbody>
</table>

Programming requirements

The caller can include the IHAFPRET mapping macro to get equate symbols for the return and reason codes provided by the IEAFP macro.

Restrictions

IEAFP must not be issued from an asynchronous exit routine.

Input register information

Before issuing the IEAFP macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:
Register

Contents
0  Reason code, when GPR 15 is non-zero
1  Used as a work register by the system
2-13 Unchanged
14-15 Used as a work register by the system
15  Return code

When control returns to the caller, the ARs contain:

Register

Contents
0-1 Used as work registers by the system
2-13 Unchanged
14-15 Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IEAFP macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td></td>
<td>One or more blanks must precede IEAFP.</td>
</tr>
<tr>
<td></td>
<td>One or more blanks must follow IEAFP.</td>
</tr>
<tr>
<td>STOP</td>
<td></td>
</tr>
<tr>
<td>STOPVECTOR</td>
<td></td>
</tr>
<tr>
<td>START</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

`name`
An optional symbol, starting in column 1, that is the name on the IEAFP macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

`STOP`
`STOPVECTOR`
`START`
A required input parameter that specifies the type of request.

`STOP`
Indicates to stop saving additional floating point status and vector register status until such time as new operations require it.

`STOPVECTOR`
Indicates to stop saving vector register status while continuing the saving of floating point status.

`START`
Indicates that the system is to start saving vector register status. This is intended for use primarily by preemptible SRBs but may be used by a task. A preemptible SRB that intends to use vector registers must use IEAFP START before the first vector register use.

IEAFP START is not supported for tasks that have any of the following characteristics:
- Are disabled for I/O or external interrupts
- Hold a lock other than the LOCAL lock (of the home address space)
- Have an EUT FRR

Do not issue IEAFP START under any of the following conditions:
- From a non-preemptible SRB
- From a task that is the initiator task or a sibling of that task or an ancestor of that task
- While holding any system locks other than the LOCAL lock of the home address space

The service will not succeed in the event that storage cannot be obtained.

`RETCODE=retcode`
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

`RSNCODE=rsnocode`
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

ABEND codes

None.
## Return and reason codes

When the IEAFP macro returns control to your program:

- GPR 15 (and `recode`, when you code `RETCODE`) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and `rsncode`, when you code `RSNCODE`) contains a reason code.

Macro IHAFPRET provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the `xxxx` value.

### Table 40. Return and reason codes for the IEAFP macro

<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason code</th>
<th>Equate symbol, meaning, and action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>—</td>
<td><strong>Equate symbol:</strong> <code>ieafpRc_OK</code>&lt;br&gt;<strong>Meaning:</strong> IEAFP request successful.&lt;br&gt;<strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>8</td>
<td>—</td>
<td><strong>Equate symbol:</strong> <code>ieafpRc_InvParm</code>&lt;br&gt;<strong>Meaning:</strong> IEAFP request specifies parameters that are not valid.&lt;br&gt;<strong>Action:</strong> Refer to the action provided with the specific reason code.</td>
</tr>
<tr>
<td>8</td>
<td><code>xxxx0801</code></td>
<td><strong>Equate symbol:</strong> <code>ieafpRsnBadFunction</code>&lt;br&gt;<strong>Meaning:</strong> Incorrect value passed to target routine.&lt;br&gt;<strong>Action:</strong> Check for possible storage overlay.</td>
</tr>
<tr>
<td>C</td>
<td>—</td>
<td><strong>Equate symbol:</strong> <code>ieafpRc_Env</code>&lt;br&gt;<strong>Meaning:</strong> Environmental error&lt;br&gt;<strong>Action:</strong> Refer to the action provided with the specific reason code.</td>
</tr>
<tr>
<td>C</td>
<td><code>xxxx0C01</code></td>
<td><strong>Equate symbol:</strong> <code>ieafpRsnFromAsynchExit</code>&lt;br&gt;<strong>Meaning:</strong> IEAFP was issued from an asynchronous exit routine.&lt;br&gt;<strong>Action:</strong> Avoid issuing IEAFP from an asynchronous exit routine.</td>
</tr>
<tr>
<td>C</td>
<td><code>xxxx0C02</code></td>
<td><strong>Equate symbol:</strong> <code>ieafpRsnFromNonPreemptibleSRB</code>&lt;br&gt;<strong>Meaning:</strong> IEAFP START was issued from an SRB that was a non-preemptible SRB.&lt;br&gt;<strong>Action:</strong> Avoid issuing IEAFP from a non-preemptible SRB.</td>
</tr>
<tr>
<td>C</td>
<td><code>xxxx0C03</code></td>
<td><strong>Equate symbol:</strong> <code>ieafpRsnNotFromBITCB</code>&lt;br&gt;<strong>Meaning:</strong> IEAFP START was issued from a task that was not the jobstep program task nor a subtask of that task.&lt;br&gt;<strong>Action:</strong> When using IEAFP START from a task, do so only from the jobstep program task or a subtask of that task.</td>
</tr>
</tbody>
</table>
### Table 40. Return and reason codes for the IEAFP macro (continued)

<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason code</th>
<th>Equate symbol, meaning, and action</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>xxxC04</td>
<td><strong>Equate symbol</strong>: ieafpRsnLocked</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: IEAFP START was issued while holding a system lock other than the LOCAL lock.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Use IEAFP START only when holding no system lock or when holding only the LOCAL lock.</td>
</tr>
<tr>
<td>C</td>
<td>xxxC05</td>
<td><strong>Equate symbol</strong>: ieafpRsnNoStorage</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: Necessary system storage could not be obtained.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Use IEAFP START at an earlier time in the jobstep.</td>
</tr>
</tbody>
</table>

### Example

**Operation:**

1. Stop additional status saving.

The code is as follows:

```plaintext
IEAFP STOP
```
Chapter 34. IEALSQRY — Linkage stack query

Description

The linkage stack query macro IEALSQRY checks the level of the current entry on the linkage stack relative to the level of the entry associated with the most recent recovery routine. The output of the macro is a value (in the TOKEN parameter) a recovery routine can use to ensure that a retry routine runs with the appropriate linkage stack entry. If the return code is not zero, the value in TOKEN is not valid.

Your program is to pass the value in TOKEN to a recovery routine. When the recovery routine gets control, it can place that value in the SDWA field SDWALSLV. That action ensures that, when a retry routine gets control, it has the correct linkage stack level. For information about how to use the value in TOKEN, see the topic about the linkage stack at a retry routine in z/OS MVS Programming: Authorized Assembler Services Guide.

The output of IEALSQRY depends upon the current environment and on the recovery environment that exists:

- If FRRs exist, the value returned in TOKEN is the difference between the current level of the linkage stack and the level of the stack at the time the FRR was activated.
- If no FRRs exist, but the caller holds a lock or is in SRB mode, a return code of 8 is returned.
- If no FRRs exist, and the caller is unlocked and in task mode, and at least one ESTAE-type recovery routine is in effect, the output depends on the most recently activated routine:
  - If it is a STAE, STAI, or FESTAE routine, a return code of 8 is returned.
  - If it is an ARR, the value returned in TOKEN is the difference between the current level of the linkage stack and the level of the stack at the time the ARR was activated.
  - If it is an ESTAE or ESTAEX for the current RB, the value returned is the difference between the current level of the linkage stack and the level of the stack at the time the ESTAE or ESTAEX was activated.
  - If it is an ESTAI, the value returned is the difference between the current level of the linkage stack and the level of the stack at the time the newest PRB that is older than the oldest non-PRB was created (or simply the newest PRB if all the RBs are PRBs).
- If no FRRs exist, and the caller is unlocked and in task mode, and no ESTAEXs, ESTAEs, STAEs, or FESTAEs exist for this RB and no ESTAIs, STAIIs, or ARRs in effect, a return code of 8 is returned.

See z/OS MVS Programming: Authorized Assembler Services Guide for further information about the use of the SDWALSLV field.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state, PSW key 8-15</td>
</tr>
</tbody>
</table>
IEALSQRY macro

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>Amode:</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller can hold the local lock of the primary address space and can additionally hold the CMS lock. The caller can hold the CPU lock. No locks are required. If the primary address space does not match the home address space, the caller must not hold the local lock of the home address space.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements
None.

Restrictions
Do not issue the IEALSQRY macro in a DIE routine.

Input register information
Before issuing the IEALSQRY macro, the caller does not have to place any information into a general purpose register (GPR) or access register (AR).

Output register information
When control returns to the caller from IEALSQRY, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Output token value, which is copied to the area specified by the TOKEN parameter.</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Return code.</td>
</tr>
</tbody>
</table>

When control returns to the caller from IEALSQRY, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14 and 15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

Performance implications
This macro should not be used in a performance-sensitive program.
Syntax

The standard form of the IEALSQRY macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEALSQRY.</td>
</tr>
<tr>
<td>IEALSQRY</td>
<td>One or more blanks must follow IEALSQRY.</td>
</tr>
<tr>
<td>TOKEN=token</td>
<td>token: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td></td>
<td>Default: Leave token in GPR 0.</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address, or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td>Default: No retcode processing.</td>
</tr>
</tbody>
</table>

The parameters are explained as follows:

**TOKEN=token**

Specifies a halfword area (or the address of the area in register (1)-(12)) where the system places a value that indicates the difference between the number of linkage stack entries present when the recovery routine was activated and the number that are currently present. A recovery routine can place this value in field SDWALSLV (in mapping macro IHASDWA) to ensure that the retry routine runs with the proper level of the linkage stack. If you do not use TOKEN, you can find the value in GPR 0.

**RETCODE=retcode**

Specifies a fullword output variable (or register (2)-(12)) into which the system copies the return code GPR 15. If you do not use RETCODE, you can find the return code in GPR 15.

**ABEND codes**

The IEALSQRY caller might receive abend code X'B78'. For detailed abend code information, see [z/OS MVS System Codes](https://www.ibm.com/support/knowledgecenter/SSEPGG_2.2.0/com.ibm.zos.v2r2.racf/abendcode/racf_abendcode.htm).

**Return codes**

When control returns to the caller, register 15 contains one of the following decimal return codes (hexadecimal values are shown in parentheses):
### IEALSQRY macro

#### Table 41. Return Codes for IEALSQRY

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
</table>
| 0 (0)       | **Meaning:** Successful completion. A valid value is in the TOKEN parameter.  
**Action:** None required.                           |                                                                                              |
| 4 (4)       | **Meaning:** The system encountered a linkage stack entry that violates the authorization or stacking-PC conditions that are required for successful retry.  
**Action:** Avoid using the token when retrying. You cannot retry to the current linkage stack level. |                                                                                              |
| 8 (8)       | **Meaning:** No recovery routine of the proper type exists. If in a state from which you cannot issue ESTAEX, no FRR exists. If in a state from which you can issue ESTAEX, either no recovery routine exists or the most recently activated recovery routine is STAE, STAI, or FESTAE.  
**Action:** Avoid using the token when retrying. You cannot retry to the current linkage stack level. |                                                                                              |
| 12 (C)      | **Meaning:** You called IEALSQRY in a DIE routine.                       
**Action:** Do not use the IEALSQRY macro in a DIE routine.                                         |                                                                                              |
| 16 (10)     | **Meaning:** System error.                                              
**Action:** Report the problem to IBM. Avoid using the token when retrying. You cannot retry to the current linkage stack level. |                                                                                              |

#### Example

Obtain the value that a recovery routine can place in SDWALSLV:

```
IEALSQRY TOKEN=MYTOKEN
```

```
MYTOKEN DS H  Output TOKEN
```
Chapter 35. IEAMETR — Query external time reference status

Description

IEAMETR can be used to query external time reference (ETR) status and connection information for the current MVS image. This information is returned in the output area specified by the OUTAREA keyword.

Environment

The requirements for the caller are:

- **Environmental factor Requirement**: Any state or key
- **Minimum authorization**: Either Task, SRB, or DIE mode
- **Dispatchable unit mode**: Any PASN, any HASN, any SASN
- **AMODE**: 31-bit
- **ASC mode**: Primary or access register
- **Interrupt status**: Enabled or disabled for I/O and external interrupts
- **Locks**: Any lock may be held
- **Control parameters**: Must be in the primary address space

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the IEAMETR macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the IEAMETR macro, the caller does not have to place any information into any AR unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:
Register

Contents

0-1  Used as work registers by the system
2-13  Unchanged
14-15  Used as work registers by the system

Performance implications

None.

Syntax

The IEAMETR macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xlabel</td>
<td>xlabel: Optional symbol. Begin xlabel in column 1. The name must conform</td>
</tr>
<tr>
<td></td>
<td>to the rules for an ordinary assembler language symbol. DEFAULT: No name.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEAMETR.</td>
</tr>
<tr>
<td>IEAMETR</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEAMETR.</td>
</tr>
<tr>
<td>OUTADDR=xoutaddr</td>
<td></td>
</tr>
<tr>
<td>,MF=S</td>
<td>Default: S</td>
</tr>
<tr>
<td>,MF=(L,xmfctrl,xmfattr</td>
<td>0D)</td>
</tr>
<tr>
<td>,MF=(E,xmfctrl,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

OUTADDR=xoutaddr

A required input parameter that contains the address of the 24-byte output area to receive the output. The area is mapped by macro IHAETRI.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MF=S | L | E

Optional keyword input that specifies the macro form.

S  Specifies the standard form of the macro. Generates code to put the parameters into an in-line parameter list and invoke the desired service. Full checking for required macro keys is done along with supplying defaults for omitted optional parameters.

DEFAULT: S
IEAMETR macro

L  Specifies the list form of the macro. Defines an area to be used for the parameter list. Any other macro parameters are flagged as errors.

E  Specifies the execute form of the macro. Generates code to put the parameters into the parameter list specified by xmfctrl and provides syntax checking with default setting.

,xmfctrl
    Required input. It is the name of a storage for the parameter list.

,xmfattr | 0D
    Optional 60 character input string that varies from 1 to 60 characters. It can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list.

    DEFAULT: 0D. 0D forces the parameter list to a doubleword boundary.

,xmfctrl
    Required input. It is the name (RS-type), or address in register (1)-(12), of a storage area for the parameter list.

,COMPLETE
    Optional keyword input that specifies the degree of macro parameter syntax checking.

    DEFAULT: COMPLETE. Checking for required macro keys is done, and defaults are supplied for omitted optional parameters.

Return codes

Table 42. Return Codes for the IEAMETR Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                     | **Meaning:** ETR status and port data was successfully obtained.  
                         | **Action:** None. |
| 04                     | **Meaning:** ETR status information is available, but port is not.  
                         | **Action:** None. |
| 08                     | **Meaning:** No status or port data is available.  
                         | **Action:** None. |
| 0C                     | **Meaning:** The parameter list is not in the user's primary address space.  
                         | **Action:** Use a parameter list in the primary address space. |
IEAMETR macro
Chapter 36. IEAMRMF3 — Obtain address space dispatchability data

Description

The IEAMRMF3 macro provides information about the dispatchability of address spaces. Use IEAMRMF3 to determine which address spaces are currently running on a processor and which address spaces are waiting for a processor. To get information about the dispatchability of enclaves, use the IMWRQRY macro. IMWRQRY is described in [z/OS MVS Programming: Workload Management Services](#).

The output you receive from this macro contains an array of elements, with each element representing an address space. For each address space, the system indicates that the address space is one of the following:

- Dispatchable and running on a processor
- Dispatchable and waiting to run on a processor
- Not dispatchable.

The number of elements you receive is the maximum number of address spaces in the system.

Use the IEAMRMF3 macro for monitoring your system. Typically, a monitoring program issues the macro repeatedly to obtain samples over a period of time. For some invocations of the macro, the system might be unable to retrieve the data, and the caller receives a return code of 4. However, this is generally a temporary condition; if the caller was issuing the macro repeatedly, the caller should continue to do so and should receive data on subsequent invocations. If the caller receives return code 4 several times in succession, the caller should stop issuing the macro. How many times the caller issues the macro after a return code of 4 is up to the installation.

Under certain conditions, the system abnormally ends the caller of IEAMRMF3 with an X'0C4' abend code. The caller must supply its own recovery routine to capture this abend code and retry.

Environment

Requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state and PSW key 0</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN or PASN&lt;=&gt;HASN&lt;=&gt;SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held, except the CPU lock if the caller is disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space</td>
</tr>
</tbody>
</table>
Programming requirements

The caller must obtain storage for the output returned by this macro. See the OUTAREA parameter for further information.

Include the following mapping macros in the module that calculates the size of the storage area for the output:

- IHADSD, which maps the DSD data area
- IHAASVT, which maps the ASVT data area
- CVT, which maps the CVT data area
- IHAPSA, which maps the PSA data area

If a separate module examines the data returned by IEAMRMF3, that module must also include the IHADSD mapping macro.

For the mappings provided by the IHAASVT, CVT, IHADSD, and IHAPSA mapping macros, see the information in z/OS MVS Data Areas in the z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/).

Restrictions

None.

Register information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the caller issued the macro. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

On entry to IEAMRMF3, general purpose register (GPR) 13 must contain the address of a 72-byte standard save area. If the caller is disabled, the save area must be pagefixed.

When control is returned to the calling program, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control is returned to the calling program, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
Performance implications

None.

Syntax

The standard form of the IEAMRMF3 macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEAMRMF3.</td>
</tr>
<tr>
<td>IEAMRMF3</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEAMRMF3.</td>
</tr>
<tr>
<td>OUTAREA=area name</td>
<td>area name: Symbol.</td>
</tr>
</tbody>
</table>

Parameters

The parameter is explained as follows:

**OUTAREA=area name**

The required parameter that specifies the name of the area of storage to contain the output from the macro. The output contains an array of elements; each element represents an address space. The output is mapped by the IHADSD mapping macro.

The caller must obtain storage for area name on a fullword boundary, in the caller's primary address space. The caller is not required to initialize area name. If the caller is disabled, area name must be pagefixed.

Before issuing IEAMRMF3, you need to determine the size of the output area and obtain storage for it. To do so, use the following formula to determine the length of an element:

\[
(DSDFXED - DSDELEM) + (\text{length of DSDFXED})
\]

Then, use that value in the following formula to determine the total size of area name:

\[
(ASVTMAXU \times \text{length of an element}) + (\text{length of DSDFXED})
\]

ASVTMAXU is a field in the ASVT data area. DSDFXED, DSDFXED, and DSDELEM are fields in the DSD data area.

For each element representing an address space, the following are true:

- If the DSDUSING bit in the DSD is on, the address space is dispatchable and running on a processor.
- If the DSDWAIT bit in the DSD is on, the address space is dispatchable and is waiting to run on a processor.
- If neither the DSDUSING bit nor the DSDWAIT bit in the DSD is on, then either the address space is not dispatchable or the element does not represent a valid address space.
IEAMRMF3 macro

Return codes

When control returns from IEAMRMF3, GPR 15 contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: Data successfully collected.</td>
</tr>
<tr>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: The system was not able to gather the data on this invocation of the macro.</td>
</tr>
<tr>
<td></td>
<td>Action: Reissue the macro. If you receive return code 4 several times in succession, you need to stop issuing the macro and inform your technical support personnel.</td>
</tr>
</tbody>
</table>

Example

Issue the IEAMRMF3 macro to obtain address space dispatchability data. The caller in this example is enabled for I/O and external interrupts, and is APF-authorized.

In this example, the caller issues IEAMRMF3 only once. If the return code from IEAMRMF3 is zero, the caller loops through the elements to look at the data, and does not issue the macro again. If the return code is not zero, the caller does not make another attempt to obtain data. A more typical scenario would be to issue the macro repeatedly if the return code is zero to obtain data over a period of time, and to issue the macro again even if the return code is 4, hoping to obtain data on a subsequent invocation. This example is intended only as an illustration of how to issue the macro, and an illustration of one way you might look at the data returned by the macro.

```
MONITOR CSECT
MONITOR AMODE 31
MONITOR RMODE ANY
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
*
* USING *,R15            Establish addressability
* STM R14,R12,12(R13)    Save registers in caller's save area
* DROP R15               Copy base register R12 because R15 is volatile across interfaces used in this module
* LR R12,R15
* @PSTART EQU MONITOR    Reestablish addressability using
USING @PSTART,R12
```
R12 as base register

Invoke the IEAMRMF3 service to obtain address space dispatchability data.

Change to key 0, supervisor state

MODESET KEY=ZERO,MODE=SUP

Obtain storage for a register save area to pass to IEAMRMF3

LA R0,72
GETMAIN RU,LV=(R0)
LR R4,R13 Save the caller's save area address
LR R13,R1 Get address of save area to pass to IEAMRMF3

Calculate the size of the storage to obtain for the output from IEAMRMF3:

(ASVTMAXU * length of an element) + (length of DSDFIXED)

USING PSA,0
L R2,FLCCVT Get pointer to the CVT from the PSA
L R9,CVTASVT-CVTMAP(,R2) Get pointer to the ASVT from the CVT
L R9,ASVTMAXU-ASVT(,R9) Get ASVTMAXU
LA R11,ELEMSIZE Get the length of an element
MR R8,R11 Multiply
LA R6,L'DSDFIXED Get the length of DSDFIXED
AR R9,R6 Add to get total

Obtain storage for the DSD data area

GETMAIN RU,LV=(R9)

Issue the IEAMRMF3 macro to return address space dispatchability data
Note: Register 13 contains the address of the 72-byte save area to pass to IEAMRMF3.

LR R6,R1 Move address of storage that was just obtained into R6.

USING DSD,R6 Map the DSD on the storage area that was just obtained.

IEAMRMF3 OUTAREA=DSD Issue the IEAMRMF3 macro passing the DSD data area to be used for the output.

LTR R15,R15 Check return code from IEAMRMF3.
BNZ NODATA For a nonzero return code, do not attempt to look at data.

Look at the elements that are filled in by using the DSDDINDXF field to find the first element, and the DSDDINDXN field to chain to the next element.

LH R11,DSDDINDXF Get the index of the first entry that is filled in. If the value is x'FFFF' then no entries are filled in.
L R7,DSDDAPTR Get the address of the array
IEAMRMF3 macro

NEXTELEM DS 0H
   CH R11,X'FFFF'
   * If the index is X'FFFF' then
     * this is the last element that is
     * filled in.
   * BE ALLDONE
     * There are no more elements to
     * process.
   * BCTR R11,0
     * Decrement the index by 1. The
     * entry for ASID 1 is the first
     * entry.
   * LA R2,ELEMENTS
     * Get the element size
   MR R10,R2
     * Multiply the index by the
     * element size.
   * AR R11,R7
     * Add the array pointer and the
     * result to obtain the address of
     * the entry that we want to look at
   * USING DSDELEM,R11
     * .
     * .
     * .
   * LH R11,DSDINDEX
     * Obtain the index of the next
     * entry to look at.
   * DROP R11
     * Go process the next element
   * NODATA DS 0H
   LR R2,R15
     * Save the nonzero return code
     * from IEAMRMF3 in R2. R15 is
     * volatile across the interfaces.
   * B FREESTOR
   ALLDONE DS 0H
   * .
   * Set a return code of zero.
   * LA R2,0
     * Save the zero return code that
     * this module sets in R2. R15 is
     * volatile across the interfaces.
   * FREESTOR DS 0H
   * .
   * Free the storage for the register save area passed to IEAMRMF3.
   * LA R0,72
     * FREEMAIN RU,LV=(R0),A=(13)
   * .
   * Restore the caller's save area address
   * LR R13,R4
   * .
   * Free the storage for the DSD data area
   * FREEMAIN RU,LV=(R9),A=(6)
   DROP R6
     * Drop addressability to the
     * DSD data area.
   * .
   * Change to problem state, not key 0
   * MODESET KEY=NZERO,MODE=PROB
   * LR R15,R2
     * Copy the return code to
     * R15.
   L R14,12(R13)
   LM R0,R12,20(R13)
   BR R14
     * Return to the caller
   DROP R12
   * .
   * Equate for length of an element:
IEAMRF3 macro

* ELEMSIZE EQU DSDRSVD-DSDELEM+$'DSDRSVD
EJECT
IHAASVT LIST=NO Mapping macro for the ASVT
EJECT
CVT DSEC=YES,LIST=NO Mapping macro for the CVT
EJECT
IHAASD LIST=YES Mapping macro for the DSD
EJECT
IHAPSA LIST=NO Mapping macro for the PSA
END MONITOR
IEAMRMF3 macro
Chapter 37. IEAMSCHD — Schedule an SRB

Description

Use the IEAMSCHD macro to schedule a service request block (SRB) for asynchronous execution. When you schedule an SRB, you can specify dispatching priority and processor affinity. Preemptable SRBs (PRIORITY=CLIENT, PRIORITY=ENCLAVE, or PRIORITY=PREEMPT) can also be scheduled with a minor priority.

Optionally, the scheduling program can specify:

* A functional recovery routine (FRR)
* A resource manager termination routine (RMTR)

The scheduling program can specify an RMTR to be invoked by the PURGEDQ service. The RMTR is responsible for cleaning up resources on behalf of an SRB routine if it has been purged by PURGEDQ before it is dispatched.

IBM recommends using IEAMSCHD rather than the SCHEDULE macro. For information about how to schedule an SRB, see z/OS MVS Programming: Authorized Assembler Services Guide.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>PSW key 0-7, or supervisor state with any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any HASN, any PASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary, or access register</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller may hold locks, but is not required to hold any. A caller who specifies SYNCH=YES cannot hold any locks.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

* If the caller specifies RMTRADDR, the resource manager termination routine must reside in MVS common storage.
* If the caller specifies PRIORITY=ENCLAVE, the enclave token must have been previously obtained through the IWMECREA macro.

Restrictions

* Address space resource managers cannot use the STOKEN parameter.
* If you issue IEAMSCHD from a set DIE routine, you cannot specify PRIORITY=CURRENT, you cannot specify SYNCH=YES, and the DIE routine must be running in supervisor state, with PSW key 0.
* If your program specifies SYNCH=YES and the scheduled SRB issues the SRBSTAT SAVE macro or invokes any services that issue SRBSTAT SAVE, control returns to your program immediately.
IEAMSCHD macro

Input register information
Before issuing the IEAMSCHD macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
The standard form of the IEAMSCHD macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEAMSCHD macro.</td>
</tr>
<tr>
<td>IEAMSCHD</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEAMSCHD macro.</td>
</tr>
<tr>
<td>EPADDR=epaddr</td>
<td>epaddr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ENV=HOME</td>
<td>Default: HOME</td>
</tr>
<tr>
<td>Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>,ENV=PRIMARY</td>
<td>ENV=PRIMARY</td>
</tr>
<tr>
<td>,ENV=FULLXM</td>
<td>ENV=FULLXM</td>
</tr>
<tr>
<td>,ENV=STOKEN</td>
<td>ENV=STOKEN</td>
</tr>
<tr>
<td>,TARGETSTOKEN=targetstoken</td>
<td>targetstoken: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,FEATURE=NONE</td>
<td>Default: NONE</td>
</tr>
<tr>
<td>,FEATURE=Crypto</td>
<td>,FEATURE=Crypto</td>
</tr>
<tr>
<td>,FEATURE=CPMASK</td>
<td>,FEATURE=CPMASK</td>
</tr>
<tr>
<td>,PRIORITY=LOCAL</td>
<td>Default: LOCAL</td>
</tr>
<tr>
<td>,PRIORITY=GLOBAL</td>
<td>,PRIORITY=GLOBAL</td>
</tr>
<tr>
<td>,PRIORITY=CURRENT</td>
<td>,PRIORITY=CURRENT</td>
</tr>
<tr>
<td>,PRIORITY=PREEEMPT</td>
<td>,PRIORITY=PREEEMPT</td>
</tr>
<tr>
<td>,PRIORITY=CLIENT</td>
<td>,PRIORITY=CLIENT</td>
</tr>
<tr>
<td>,PRIORITY=ENCLAVE</td>
<td>,PRIORITY=ENCLAVE</td>
</tr>
<tr>
<td>,MINORPRIORITY=ZERO</td>
<td>Default: ZERO</td>
</tr>
<tr>
<td>,MINORPRIORITY=minorpriority</td>
<td>minorpriority: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,SRBIDTOKEN=token</td>
<td>token: 16-byte output area.</td>
</tr>
<tr>
<td>,CLIENTSTOKEN=clientstoken</td>
<td>clientstoken: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ENCLAVETOKEN=enclavetoken</td>
<td>enclavetoken: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PARM=ZERO</td>
<td>Default: ZERO</td>
</tr>
<tr>
<td>,PARM=parm</td>
<td>parm: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,FRRADDR=NOFRR</td>
<td>Default: NOFRR</td>
</tr>
<tr>
<td>,FRRADDR=frraddr</td>
<td>frraddr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,SDWALOC31=NO</td>
<td>Default: NO</td>
</tr>
<tr>
<td>,SDWALOC31=YES</td>
<td>,SDWALOC31=YES</td>
</tr>
<tr>
<td>,KEYVALUE=INVOKERKEY</td>
<td>Default: INVOKERKEY</td>
</tr>
<tr>
<td>,KEYVALUE=keyvalue</td>
<td>keyvalue: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,LLOCK=NO</td>
<td>Default: NO</td>
</tr>
<tr>
<td>,LLOCK=YES</td>
<td>,LLOCK=YES</td>
</tr>
</tbody>
</table>
### IEAMSCHD macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,RMTRADDR=NORMTR</td>
<td>Default: NORMTR</td>
</tr>
<tr>
<td>,RMTRADDR=rmtraddr</td>
<td>rmtraddr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PURGESTOKEN=NOPSTOKEN</td>
<td>Default: NOPSTOKEN</td>
</tr>
<tr>
<td>,PURGESTOKEN=purgestoken</td>
<td>purgestoken: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PTCBADDR=NOPTCB</td>
<td>Default: NOPTCB</td>
</tr>
<tr>
<td>,PTCBADDR=ptcbaddr</td>
<td>ptcbaddr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,FLAGS=NO_FLAGS</td>
<td>Default: NO_FLAGS</td>
</tr>
<tr>
<td>,FLAGS=flags</td>
<td>flags: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,SYNCH=NO</td>
<td>Default: NO</td>
</tr>
<tr>
<td>,SYNCH=YES</td>
<td></td>
</tr>
<tr>
<td>,SYNCHCOMPADDR=NOVALUE</td>
<td>Default: NOVALUE</td>
</tr>
<tr>
<td>,SYNCHCOMPADDR=compaddr</td>
<td>compaddr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,SYNCHCODEADDR=codeaddr</td>
<td>codeaddr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,SYNCHRSNADDR=rsnaddr</td>
<td>rsnaddr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=0</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=1</td>
<td></td>
</tr>
<tr>
<td>,MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td>list addr: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td>,MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,O0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:
EPADDR=epaddr
  Specifies the address of the SRB routine to be scheduled for asynchronous
  execution.

  **Note:** The SRB routine receives control in 31–bit addressing mode.

  **To code:** Specify the name (RS-type), or address in register (2)-(12), of a
  required 4-byte input parameter.

,ENV=HOME
,ENV=PRIMARY
,ENV=FULLXM
,ENV=STOKEN
  Optional input parameter that specifies the addressing and cross memory
  environment in which the SRB routine is to receive control.

  **HOME**
  Specifies that the SRB routine is to receive control in the current home
  address space.

  **Default:** HOME

  **PRIMARY**
  Specifies that the SRB routine is to receive control in the current primary
  address space.

  **FULLXM**
  Specifies that the SRB routine is to receive:
  · Control in the scheduling program’s current cross memory environment
  · A copy of the scheduling program’s dispatchable unit-access list
    (DU-AL). For details about how the system copies a DU-AL, see the
    topic on access lists in *z/OS MVS Extended Addressability Guide*.

  This provides the SRB routine with addressability to the same address
  spaces and data spaces as the scheduling program.

  **STOKEN**
  Specifies that the SRB routine is to receive control in the address space
  specified by TARGETSTOKEN=TARGETSTOKEN. If the target stoken passed is
  no longer valid, then the caller receives AC7 abend code with reason code
  X’00080001’.

,TARGETSTOKEN=targetstoken
  Specifies the space token (STOKEN) of the address space in which the SRB
  routine is to receive control.

  **To code:** Specify the name (RS-type), or address in register (2)-(12), of a
  required 64-bit input parameter.

,PRIORITY=LOCAL
,PRIORITY=GLOBAL
,PRIORITY=CURRENT
,PRIORITY=PREEMPT
,PRIORITY=CLIENT
,PRIORITY=ENCLAVE
  Optional input parameter that specifies the priority at which the SRB routine is
  dispatched, and whether the SRB is to be preempted.

  **LOCAL**
  Schedules an SRB at a priority equal to that of the address space into
which it was scheduled. With a LOCAL priority, an SRB has a higher priority than any task or preemptable SRB in that address space.

Default: LOCAL

GLOBAL
Schedules an SRB at a priority equal to the highest priority work in the system, regardless of the address space into which it was scheduled. An SRB scheduled with PRIORITY=GLOBAL is not preemptable.

CURRENT
Schedule an SRB at a priority equal to that of the scheduling work unit.

Task Mode Callers: For task mode callers, the SRB is always preemptable. If the task has joined an enclave, the SRB routine inherits the enclave’s major priority and the task’s minor priority. Otherwise, the SRB routine inherits the major priority of the task’s home address space and the minor priority of the task. If the scheduling task and the scheduled SRB have different home address spaces, then the scheduled SRB is also converted to a client SRB.

Nonpreemptable SRB Mode Callers: For SRB mode callers that are not preemptable, the scheduled SRB inherits the PRIORITY option used to schedule the scheduling SRB routine:

• If PRIORITY=GLOBAL was used, the scheduled SRB will have a priority as high as the highest priority in the system.
• If PRIORITY=LOCAL was used, the scheduled SRB will have a priority that is higher than any task or preemptable SRB in the scheduled SRB’s home address space.

Preemptable SRB Mode Callers: For preemptable SRB mode callers, the scheduled SRB is always preemptable. If the scheduling SRB was scheduled into an enclave, the scheduled SRB inherits the enclave’s major priority and the scheduling SRB’s minor priority. Otherwise, the scheduled SRB inherits the major priority of the scheduling SRB’s home address space and the minor priority of the scheduling SRB. If the scheduling SRB and the scheduled SRB have different home address spaces, then the scheduled SRB is also converted to a client SRB.

PREEMPT
Schedules a preemptable SRB routine that inherits the major priority of the target home address space (the home address space as specified on the ENV parameter).

CLIENT
Schedules a preemptable SRB that inherits the major priority of the address space named by the STOKEN specified on the CLIENTSTOKEN parameter. The processor time used by this SRB is accumulated in the address space specified by the clientstoken.

ENCLAVE
Schedules a preemptable RB into an enclave. The SRB inherits the major priority of the enclave specified on the ENCLAVETOKEN keyword. The processor time used by this SRB is accumulated in the enclave specified by the enclavetoken.

,MINORPRIORITY=ZERO
,MINORPRIORITY=minorpriority
Specifies the minor priority to assign to the SRB routine. SRB routines with higher minor priority are dispatched before preemptable-class SRB routines.
IEAMSCHD macro

and before tasks with lower minor priority in the same address space. A minor priority of X'00' is the lowest and X'FF' is the highest.

The minor priority parameter assigns the SRB routine a priority that is comparable to a task’s dispatching priority in the address space. The caller can specify priorities for SRB routines so that they are dispatched before, with, or after tasks in the address space.

Default: ZERO

To code: Specify the name (RS-type), or address in register (2)-(12), of an 8-bit input parameter. MINORPRIORITY is optional for PRIORITY=PREEMPT, PRIORITY=CLIENT, and PRIORITY=ENCLAVE.

\texttt{,SRBIDTOKEN=token}

Specifies the name of an optional 16-byte output area where a token is placed to be used to fully identify the SRB to the system. The token is used to request termination of a preemptable SRB via CALLRTM TYPE=SRBTERM. The SRBIDTOKEN keyword may be used even if the program runs on a release for which the support is not provided or on a release on which the support is not installed. If the program is running on a release that supports SRBIDTOKEN, the returned SRBIDTOKEN will have a non-zero value in the first eight bytes.

SRBIDTOKEN is optional for PRIORITY=PREEMPT, PRIORITY=CLIENT, and PRIORITY=ENCLAVE.

\texttt{,CLIENTTOKEN=clientstoken}

Specifies the space token (STOKEN) of the address space where the processor time used by the SRB is to be accumulated. The SRB also inherits the major priority of this address space. This parameter is a required input parameter for PRIORITY=CLIENT.

To code: Specify the name (RS-type), or address in register (2)-(12), of a required 64-bit parameter.

\texttt{,ENCLAVETOKEN=enclavetoken}

Specifies the enclave token representing the group of SRB routines. The enclave token must be obtained prior to scheduling the SRB.

To code: Specify the name (RS-type), or address in register (2)-(12), of an 8-character input parameter. ENCLAVETOKEN=enclavetoken is required for PRIORITY=ENCLAVE.

\texttt{,FEATURE=NONE,FEATURE=CRYPTO,FEATURE=CPMASK}

Optional parameter that specifies affinity to specific processors.

NONE

Specifies that there is no affinity to specific processors.

Default: NONE

CRYPTO

Specifies that the SRB routine must run on a processor that has an Integrated Cryptographic Feature (ICRF) associated with it. When you specify this parameter, the system assigns the correct processor affinity for the SRB routine. Use FEATURE=CRYPTO only for SRB routines whose exclusive purpose is to encrypt or decrypt data.

LLOCK=NO
**IEAMSCHD macro**

**LLOCK=**YES  
Specifies whether the SRB is to receive control with the LOCAL lock held. The LOCAL lock is the lock of the home address space.  
Default: NO

**FRRADDR=**NOFRR  
**FRRADDR=**fraddr  
Specifies the name (RS-type), or address in register (2)-(12), of an optional 4 byte input that contains the address of the Functional Recovery Routine (FRR) that is to be established prior to the SRB routine receiving control. The low bit of this address should not be set on. If it is set on, that bit will not be treated as part of the FRR address, but will be treated as indicating SDWALOC31=YES and will override the specification, or default, of SDWALOC31=NO.  
The FRR receives control in supervisor state, PSW key 0, primary ASC mode, 31-bit addressing mode, holding the same locks the SRB routine held at the time of error. The FRR receives control with the same PASID, SASID, and HASID as the SRB routine had on entry.  
If you specify LLOCK=YES, then the FRR should release the LOCAL lock prior to the completion of its processing.  
Default: NOFRR. The SRB routine will receive control without its own FRR.  
To code: Specify the name (RS-type), or address in register (2)-(12), of an optional 4-byte input parameter.

**SDWALOC31=**NO  
**SDWALOC31=**YES  
Specifies whether the FRR specified by FRRADDR can tolerate an SDWA in 31-bit addressable storage. Considering that 31-bit storage is less likely to be constrained than 24-bit storage and RTM skips FRRs for which it can not obtain an SDWA, use SDWALOC31=YES whenever possible. SDWALOC31 is valid only for FRRADDR.  
Default: NO

**KEYVALUE=**INVOKERKEY  
KEYVALUE=**keyvalue  
Specifies the name or address of an optional 8-bit input. Bits 0-3 contain the PSW key in which the SRB is to receive control. Bits 4-7 are ignored. For example, the byte required to specify PSW key 7 contains the value X'70', and the byte required to specify PSW key 11 contains the value X'BO'.  
Default: INVOKERKEY

If INVOKERKEY is not specified the SRB routine receives control with the PSW key of the invoker of the IEAMSCHD macro.  
To code: Specify the name (RS-type), or address in register (2)-(12), of an optional 8-bit input parameter.

**RMTRADDR=**NORMTR  
RMTRADDR=**rmtraddr  
Specifies the address of an SRB resource manager termination routine (RMTR). RMTRs are responsible for cleaning up resources on behalf of an SRB routine that has been purged by the PURGEDQ service before the SRB is first dispatched.  
The RMTR must reside in the MVS common area because the address space where the RMTR will get control is unpredictable at the time of the invocation of the IEAMSCHD macro. It is called from the PURGEDQ service and will
receive control in task mode, supervisor state, PSW key 0, primary ASC mode, and 31-bit AMODE. If bit 31 of the RMTRADDR is one, control is received with the local lock held and control can return to the PURGEDQ service with or without the local lock held, but must not hold any other locks upon return. If bit 31 of the RMTRADDR is zero, control is received with no locks held and control must be returned to the PURGEDQ service with no locks held. Bit 31 of RMTRADDR is treated as zero when determining the RMTR address.

**Default:** NORMTR

*To code:* Specify the name (RS-type), or address in register (2)-(12), of an optional 4-byte input parameter.

\[ \text{PARM} = \text{ZERO} \]

\[ \text{PARM} = \text{parm} \]

Specifies input to be loaded into register 1 when the SRB routine receives control.

**Default:** ZERO

*To code:* Specify the name (RS-type), or address in register (2)-(12), of a fullword input parameter.

\[ \text{PURGESTOKEN} = \text{NOPSTOKEN} \]

\[ \text{PURGESTOKEN} = \text{purgestoken} \]

Specifies the space token of an address space to be associated with this SRB routine. During memory termination, all SRB routines that are scheduled into the address space and have not received control are purged and control will be given to each SRB routine’s RMTR.

The address space represented by the purgestoken does not have to be the same as the address space where the SRB routine will be dispatched.

**Default:** NOPSTOKEN

*To code:* Specify the name (RS-type), or address in register (2)-(12), of an optional 64-bit input parameter.

\[ \text{PTCBADDR} = \text{NOPTCB} \]

\[ \text{PTCBADDR} = \text{ptcbaddr} \]

Specifies the address of the TCB to be related to the SRB routine. When a SYNCH=NO SRB routine scheduled with a related task terminates abnormally and the FRR for the SRB routine does not exist or percolates, the error is percolated to the recovery routine of the related task. This is known as SRB-to-task percolation.

If you specify PTCBADDR, then you must specify PURGESTOKEN.

**Default:** NOPTCB

*To code:* Specify the name (RS-type), or address in register (2)-(12), of an optional 4-byte input parameter.

\[ \text{FLAGS} = \text{NO_FLAGS} \]

\[ \text{FLAGS} = \text{flags} \]

Specifies an optional 1-byte input/output field that provides information about the scheduling of the SRB. The caller must zero this byte before invoking IEAMSCHD.

- When bit 7 (the x’01’ bit) is on, the SRB has been successfully scheduled.
- When bit 7 is off, but bit 6 (the x’02’ bit) is on, the system had just begun the final part of the scheduling of the SRB, but that did not complete successfully; it is not known whether the SRB will or will not get control.
When both bits 6 and 7 are off, the SRB was not successfully scheduled. This will be an extremely rare circumstance.

These bit definitions are true whether control returns normally to the caller or whether control passes to the caller's recovery. The byte must be in disabled-reference or page-fixed storage.

**Default:** NO_FLAGS

**To code:** Specify the name (RS-type), or address in register (2)-(12), of an optional 1-byte input output parameter.

```
\SYNCH=NO
\SYNCH=YES
```

Specifies whether or not the caller's work unit is to be suspended until the scheduled SRB completes, is purged, or ends abnormally:

**SYNCH=NO**

The SRB is to be scheduled but not synchronized with the caller's work unit.

**SYNCH=YES**

The SRB is to be scheduled and synchronized with the caller's work unit; the caller's work unit is suspended until the SRB completes, is purged, or ends abnormally. SRB to task percolation does not occur when the SRB is scheduled with the SYNCH=YES option.

**Default:** NO

```
\SYNCHCOMPADDR=NOVALUE
\SYNCHCOMPADDR=compaddr
```

When you specify SYNCH=YES, you can specify this optional parameter, which contains one of the following completion codes when the caller's work unit resumes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SRB completed successfully.</td>
</tr>
<tr>
<td>8</td>
<td>SRB ended abnormally; there is an associated reason code.</td>
</tr>
<tr>
<td>12</td>
<td>SRB ended abnormally; there is no associated reason code.</td>
</tr>
<tr>
<td>16</td>
<td>PURGEDQ processing purged the SRB.</td>
</tr>
<tr>
<td>20</td>
<td>SRB state is undetermined. It was dispatched but did not complete. A probable cause is address space termination or an error in the dynamic address translation (DAT) process.</td>
</tr>
<tr>
<td>24</td>
<td>SRB was not scheduled; SYNCHCODEADDR contains the return code from the SUSPEND service.</td>
</tr>
<tr>
<td>28</td>
<td>SRB was not scheduled; SYNCHCODEADDR contains the abend code from the SUSPEND service.</td>
</tr>
</tbody>
</table>

**Default:** NOVALUE

**To code:** Specify the name (RS-type) of an optional 4-byte input area that contains the address of the fullword that is to hold the data to be returned. When you specify this parameter, you must also specify SYNCHCODEADDR and SYNCHRSNADDR, which can provide additional information about the completion code.
When the caller's work unit resumes, contains information associated with the completion code returned through SYNCHCOMPADDR. The completion codes and the associated information are:

<table>
<thead>
<tr>
<th>Code</th>
<th>SYNCHCODEADDR Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Contents of GPR 15 when the SRB completed.</td>
</tr>
<tr>
<td>8</td>
<td>Abend code in the same format as field SDWAABCC in the SDWA.</td>
</tr>
<tr>
<td>12</td>
<td>Abend code in the same format as field SDWAABCC in the SDWA.</td>
</tr>
<tr>
<td>16</td>
<td>X'FFFFFFFF' (-1), indicating that there is no meaningful value to return.</td>
</tr>
<tr>
<td>20</td>
<td>X'FFFFFFFF' (-1), indicating that there is no meaningful value to return.</td>
</tr>
<tr>
<td>24</td>
<td>Return code from the SUSPEND service. The SRB was not scheduled because this work unit could not be successfully suspended.</td>
</tr>
<tr>
<td>28</td>
<td>Abend code from the SUSPEND service. The SRB was not scheduled because this work unit could not be successfully suspended.</td>
</tr>
</tbody>
</table>

For example, if SYNCHCOMPADDR contains a completion code of 8, then SYNCHCODEADDR contains an abend code. (If the scheduled SRB exits with the TCTL macro, SYNCHCODEADDR does not contain meaningful data; its contents are unpredictable.)

To code: Specify the name (RS-type) of an optional 4-byte input area that contains the address of the fullword that is to hold the data to be returned.

When the caller's work unit resumes, contains additional information associated with the completion code returned through SYNCHCOMPADDR and the information returned through SYNCHCODEADDR. The completion codes and the associated information are:

<table>
<thead>
<tr>
<th>Code</th>
<th>SYNCHRSNADDR Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Contents of GPR 0 when the SRB completed.</td>
</tr>
<tr>
<td>8</td>
<td>Reason code associated with an abend code.</td>
</tr>
<tr>
<td>12</td>
<td>X'FFFFFFFF' (-1), indicating that there is no meaningful value to return.</td>
</tr>
<tr>
<td>16</td>
<td>X'FFFFFFFF' (-1), indicating that there is no meaningful value to return.</td>
</tr>
<tr>
<td>20</td>
<td>X'FFFFFFFF' (-1), indicating that there is no meaningful value to return.</td>
</tr>
<tr>
<td>24</td>
<td>X'FFFFFFFF' (-1), indicating that there is no meaningful value to return.</td>
</tr>
<tr>
<td>28</td>
<td>Reason code associated with the abend code issued during an unsuccessful attempt to suspend the current work unit.</td>
</tr>
</tbody>
</table>

For example, if SYNCHCOMPADDR contains a completion code of 8, then SYNCHCODEADDR contains an abend code, and SYNCHRSNADDR contains the reason code associated with the abend code. (If the scheduled SRB exits with the TCTL macro, SYNCHCODEADDR and SYNCHRSNADDR do not contain meaningful data; the contents of both are unpredictable.)

To code: Specify the name (RS-type) of an optional 4-byte input area that contains the address of the fullword that is to hold the data to be returned.

An optional output parameter into which the return code is to be copied from GPR 15.
IEAMSCHD macro

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
,PLISTVER=1

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
  
  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, if you use only the following parameters:
  - CLIENTSTOKEN
  - ENCLAVENTOKEN
  - ENV
  - EPADDR
  - FEATURE
  - FRRADDR
  - KEYVALUE
  - LLOCK
  - MINORPRIORITY
  - PARM
  - PLISTVER
  - PRIORITY
  - PTCBADDR
  - PURGESTOKEN
  - RMTRADDR
  - TARGETSTOKEN
- **1**, if you use any of the following parameters, and parameters from version 0, or both:
  - SYNCH
  - SYNCHCODEADDR
  - SYNCHCOMPADDR
  - SYNCHRZRADDR

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
IEAMSCHD macro

- A decimal value of 0 or 1
  MF=S
  MF=(L,list addr)
  MF=(L,list addr,attr)
  MF=(L,list addr,0D)
  MF=(E,list addr)
  MF=(E,list addr,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes
IEAMSCHD might abnormally end with system completion code AC7. See z/OS MVS System Codes for an explanation and programmer responses for this code.

Return codes
When the IEAMSCHD macro returns control to your program, GPR 15 contains a return code.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: Successful completion.</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: Warning. The enclave token is not valid. The enclave token specified on the ENCLAVETOKEN parameter has been reused for a new enclave. The SRB was not scheduled.</td>
</tr>
<tr>
<td>08</td>
<td>Meaning: Program error. The client STOKEN address space has failed. The SRB was not scheduled.</td>
</tr>
</tbody>
</table>
IEAMSCHD macro

Table 44. Return Codes for the IEAMSCHD Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C</td>
<td><strong>Meaning</strong>: Program error. The purge STOKEN address space has failed. The SRB was not scheduled.</td>
</tr>
<tr>
<td>10</td>
<td><strong>Meaning</strong>: Program error. The target STOKEN address space has failed. The SRB was not scheduled.</td>
</tr>
<tr>
<td>1C</td>
<td><strong>Meaning</strong>: Program error. A SYNCH=YES SRB was not scheduled or did not complete successfully. This is set only if the SYNCHCOMPADDR parameter is used with an operand that is not NOVALUE and does not indicate that location 0 is to be where data is returned. The values returned on SYNCHCOMPADDR, SYNCHCODEADDR, and SYNCHRSNADDR contain additional information.</td>
</tr>
</tbody>
</table>

Example 1

Schedule a local SRB routine to the current home address space. The SRB routine will be entered in the same key as the scheduling program.

* 
SCHED_SRB_RTN EQU * 
IEAMSCHD EPADDR=EP_ADDR,ENV=HOME,PRIORITY=LOCAL 
* 
EP_ADDR DC A(SRB_ROUTINE) Address of Entry Point for SRB 
*

Example 2

Schedule an SRB routine to the current primary address space which has the same addressability and cross memory environment as the scheduling program and has a GLOBAL priority. The SRB routine is to receive control in PSW Key 0.

* 
SCHED_SRB_RTN EQU * 
IEAMSCHD EPADDR=EP_ADDR,ENV=FULLXM, X 
  PRIORITY=GLOBAL, X 
  KEYVALUE=PSW_KEY_0 
* 
EP_ADDR DC A(SRB_ROUTINE) Address of Entry Point for SRB 
PSW_KEY_0 DC X'00' PSW Key 0 
*

Example 3

Schedule an SRB routine at a priority that is the lowest in the enclave identified by the token in ENCLAVE_TOKEN. The SRB routine is to receive control in the current home address space with an FRR established and holding the local lock of the current home address space. It is to run in the current home address space and is to run in key 2. The SRB routine has a resource manager termination routine whose entry point address is in RMTR_ADDR. The current task's recovery is to receive control should the SRB routine's recovery percolate and the SRB routine should be purged if the current task terminates. This example assumes that ENCLAVE_TOKEN and PURGE_STOKEN were previously initialized.

* 
SCHED_SRB_RTN EQU * 
  USING PSA,0 Base Prefixed Save Area 
* 
IEAMSCHD EPADDR=EP_ADDR,FRRADDR=FRR_ADDR, X 
  KEYVALUE=PSW_KEY_2,PRIORITY=ENCLAVE, X
IEAMSCHD macro

ENCLAVE_TOKEN=ENCLAVE_TOKEN, X
MINOR_PRIORITY=MINOR_PRIORITY, X
RMTR_ADDR=RMTR_ADDR, X
PURGE_TOKEN=PURGE_TOKEN, X
PTB_ADDR=PTB_ADDR, X
LLOCK=YES, ENV=HOME

Example 4

Schedule a LOCAL SRB routine into the address space whose STOKEN is stored in THEIR_TOKEN. This example assumes that THEIR_TOKEN was previously initialized.

* SCHED_SRB_RTN EQU *
  IEAMSCHD EPADDR=EP_ADDR, ENV=STOKEN, X
  TARGETSTOKEN=THEIR_TOKEN, X
  PRIORITY=LOCAL

* .

THEIR_TOKEN DS D Space Token
EP_ADDR DC A(SRB_ROUTINE) SRB Entry Point Address

Note that in this example, the SRB routine is running in a different address space from the scheduling code. To run an SRB routine in a different address space from the scheduling code, the SRB must be either in a different program that is accessible from the target address space, or in the common storage together with the scheduling code.

Example 5

Schedule a preemtatable SRB routine into the current home address space with a minor priority that is just below the current task’s dispatching priority.

* SCHED_SRB_RTN EQU *

  EXTRACT TCB_PRIORITY,'S', FIELDS=(PRI)

* SLR 3,3 Clear register
IC 3,DSP_PRIORITY Get Dispatching Priority
S 3,=F'1' Lower priority by 1
BP SAVE_MINOR_PRIORITY
SLR 3,3 If tasks priority already lowest
* set minor priority to zero.

SAVE_MINOR_PRIORITY EQU *
STC 3,MINOR_PRIORITY Save Minor Priority
IEAMSCHD EPADDR=EP_ADDR PRIORITY=PREEMPT, ENV=HOME, X
MINOR_PRIORITY=MINOR_PRIORITY

Chapter 37. IEAMSCHD — Schedule an SRB 521
IEAMSCHD macro

Example 6
Schedule an SRB routine into the home address space, passing it the parameter list pointed to by PARM_ADDR, and give the SRB routine affinity to online processors with the Integrated Cryptographic Feature installed. The SRB routine is to inherit the current work unit's major and minor priorities.

*  SCHED_SRB_RTN  EQU *
  *  IEAMSCHD  EPADDR=EP_ADDR, PARM=PARM_ADDR, X
  *  FEATURE=CRYPTO, ENV=HOME, PRIORITY=CURRENT
  *
EP_ADDR   DC   A(SRB_ROUTINE)   SRB Entry Point Address
PARM_ADDR DC   A(PARM_LIST)   Pointer to parameter list

Example 7
Schedule a synchronous LOCAL SRB routine into the address space whose STOKEN is stored in THEIR_STOKEN. The invoker of IEAMSCHD will be suspended until the SRB routine completes, abends, or is purged. This example assumes that THEIR_TOKEN was previously initialized.

*  SCHED_SRB_RTN  EQU *
  *  IEAMSCHD  EPADDR=EP_ADDR, ENV=STOKEN, X
  *  TARGETSTOKEN=THEIR_STOKEN, PRIORITY=LOCAL, X
  *  SYNCH=YES, SYNCHCOMPADDR=COMPCODE, X
  *  SYNCHCODEADDR=ABENDCODE, SYNCHRSNADDR=REASONCODE

THEIR_STOKEN DS D    Space Token
EP_ADDR   DC   A(SRB_ROUTINE)   SRB Entry Point Address
COMPCODE   DS   F
ABENDCODE   DS   F
REASONCODE   DS   F

Note that in this example, the SRB routine is running in a different address space from the scheduling code. To run an SRB routine in a different address space from the scheduling code, the SRB routine must be either in a different program that is accessible from the target address space, or in the common storage together with the scheduling code.
Chapter 38. IEANTCR — Create a name/token pair

Description

Call the IEANTCR service to create a name/token pair.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state, with any PSW key.</td>
</tr>
<tr>
<td></td>
<td>Note: Problem-state programs with PSW key 8 - 15 cannot create system-level pairs.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td></td>
<td>Note: SRB-mode callers cannot create a task-level pair.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>The parameter list and all parameters must reside in the caller's primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

Before you use name/token services, you can optionally include the IEANTASM macro to invoke name/token services equate (EQU) statements. IEANTASM provides the following constants for use in your program:

* Name/Token Level Constants

<table>
<thead>
<tr>
<th></th>
<th>EQU</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEANT_TASK_LEVEL</td>
<td>1</td>
</tr>
<tr>
<td>IEANT_HOME_LEVEL</td>
<td>2</td>
</tr>
<tr>
<td>IEANT_PRIMARY_LEVEL</td>
<td>3</td>
</tr>
<tr>
<td>IEANT_SYSTEM_LEVEL</td>
<td>4</td>
</tr>
<tr>
<td>IEANT_TASKAUTH_LEVEL</td>
<td>11</td>
</tr>
<tr>
<td>IEANT_HOMEAUTH_LEVEL</td>
<td>12</td>
</tr>
<tr>
<td>IEANT_PRIMARYAUTH_LEVEL</td>
<td>13</td>
</tr>
</tbody>
</table>

* Name/Token Persistence Constants

<table>
<thead>
<tr>
<th></th>
<th>EQU</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEANT_NOPERSIST</td>
<td>0</td>
</tr>
<tr>
<td>IEANT_PERSIST</td>
<td>1</td>
</tr>
<tr>
<td>IEANT_NOCHECKPOINT</td>
<td>0</td>
</tr>
<tr>
<td>IEANT_CHECKPOINTOK</td>
<td>2</td>
</tr>
</tbody>
</table>

* Name/Token Return Code Constants

<table>
<thead>
<tr>
<th></th>
<th>EQU</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEANT_OK</td>
<td>0</td>
</tr>
<tr>
<td>IEANT_DUP_NAME</td>
<td>4</td>
</tr>
<tr>
<td>IEANT_NOT_FOUND</td>
<td>4</td>
</tr>
<tr>
<td>IEANT_24BITMODE</td>
<td>8</td>
</tr>
<tr>
<td>IEANT_NOT_AUTH</td>
<td>16</td>
</tr>
<tr>
<td>IEANT_SRBO_MODE</td>
<td>20</td>
</tr>
<tr>
<td>IEANT_LOCK_HELD</td>
<td>24</td>
</tr>
<tr>
<td>IEANT_LEVEL_INVALID</td>
<td>28</td>
</tr>
</tbody>
</table>
IEANTCR callable service

IEANT_NAME_INVALID EQU 32
IEANT_PERSIST_INVALID EQU 36
IEANT_AR_INVALID EQU 40
IEANT_UNEXPECTED_ERR EQU 64

Restrictions
Do not use the IEANTCR callable service in a RESMGR resource manager routine unless one of the following is true:
• The name/token pair is a system-level persistent name/token pair.
• The resource manager is running for a daughter task of the task that owns the name/token pair.
• The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.

Input register information
Before issuing the IEANTCR callable service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the general purpose registers (GPRs) contain:
Register
   Contents
0-1    Used as work registers by the system
2-13   Unchanged
14     Used as a work register by the system
15     Return code

When control returns to the caller, the access registers (ARs) contain:
Register
   Contents
0-1    Used as work registers by the system
2-13   Unchanged
14-15  Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEANTCR</td>
<td></td>
</tr>
<tr>
<td>,level</td>
<td></td>
</tr>
<tr>
<td>,user_name</td>
<td></td>
</tr>
<tr>
<td>,user_token</td>
<td></td>
</tr>
<tr>
<td>,persist_option</td>
<td></td>
</tr>
<tr>
<td>,return_code</td>
<td></td>
</tr>
</tbody>
</table>

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEANTCR:

1. **LOAD EP=IEANTCR**
   - Save the entry point address
   - (...) Put the saved entry point address into R15
   - CALL (15),(...)

2. **L 15,X'10'**
   - L 15,X'220'(15,0)
   - L 15,X'14'(15,0)
   - L 15,X'04'(15,0)
   - CALL (15),(...)

This second technique requires AMODE=31, and, before the CALL is issued, verification that the IEANTCR service is supported by the system (in the CVT, both the CVTOSEXT and the CVTOS390 bits are set on).

### Parameters

The parameters are explained as follows:

**level**

Specifies a fullword that contains an integer indicating the level of the name/token pair:

- 1 Task
- 2 Home address space
- 3 Primary address space
- 4 System.

**user_name**

Specifies the 16-byte area containing the name of the name/token pair that the user creates. The bytes of the name may have any value. The name may contain blanks, integers, or addresses.

Names must be unique within a level. Here are some examples:

- Two task-level name/token pairs owned by the same task cannot have the same name. However, two task-level name/token pairs owned by different tasks can have the same name.
- Two home-address-space-level name/token pairs in the same address space cannot have the same name. However, two home-address-space-level name/token pairs in different address spaces can have the same name.

Because of these unique requirements you must avoid using the same names that IBM uses for name/token pairs. Do not use the following names:

- Names that begin with A through I
### IEANTCR callable service

- Names that begin with X'00'.

**user_token**

Specifies the 16-byte area containing the token of the name/token pair that the user creates.

**persist_option**

Specifies a fullword that contains an integer indicating if a system-level name/token pair should persist after the creating address space's job step task terminates or if Checkpoint/Restart can be issued if the program has this task-level name/token pair. If a program has non-task-level name/token pairs or has task-level name/token pairs that did not specify IEANT_CHECKPOINTOK, the program cannot take a checkpoint.

- 0 - system-level pair will not persist and checkpoint is not permitted.
- 1 - system-level pair will persist.
- 2 - checkpoint is permitted.

**Note:** Only system-level name/token pairs can persist after the creating task terminates. Only task-level name/token pairs can permit checkpoint. You must specify 0 for all other levels.

**return_code**

Specifies a fullword to contain the return code from the IEANTCR service.

### ABEND codes

The caller might encounter abend X'AC7' with a reason code of either X'00030000' or X'00030001'. See z/OS MVS System Codes for an explanation and responses for these codes.

### Return and reason codes

When IEANTCR returns control to your program, GPR 15 and return_code contain a return code. The following table identifies return codes in hexadecimal and decimal, tells what each means, and recommends an action that you need to take:

**Table 45. Return Codes for the IEANTCR Macro**

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | Meaning: The operation was successful.  
                          | Action: None.               |
| 04                      | Meaning: The user_name specified already exists.  
                          | Action: Choose a different user_name.   |
| 08                      | Meaning: The request is rejected because the caller is in 24-bit addressing mode.  
                          | Action: Change your program to 31-bit addressing mode. |
| 10                      | Meaning: An unauthorized caller attempted to create a system-level name/token pair.  
                          | Action: Check which level of name/token pair you are creating. |
| 14                      | Meaning: An SRB-mode caller attempted to create a task-level name/token pair.  
                          | Action: Change your program to task mode or use a different level. |
| 18                      | Meaning: The caller held locks.  
                          | Action: Release all locks before issuing IEANTCR. |
### Table 45. Return Codes for the IEANTCR Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 1C                      | **Meaning:** The caller specified an incorrect level.  
Action: Respecify the correct level. Valid options are 1, 2, 3, or 4. |
| 20                      | **Meaning:** The caller specified an incorrect user_name.  
Action: Respecify the correct user_name. |
| 24                      | **Meaning:** The caller specified an incorrect persist_option.  
Action:  
- For system-level name/token pairs, you must specify zero or one for the persist_option.  
- For task-level name/token pairs, you must specify zero or two for the persist_option.  
- For home or primary address space level name/token pairs, you must specify zero for the persist_option. |
| 28                      | **Meaning:** The caller was in AR ASC mode and AR1 was not zero.  
Action: Change your program to primary mode or set AR1 to zero. |
| 40                      | **Meaning:** A system error occurred while handling the request.  
Action: Retry the request. |

### Example

Initialize the name/token fields, and create, retrieve, and delete a task-level name/token pair.

```assembly
TITLE 'NAME/TOKEN EXAMPLE PROGRAM'
NTIDSAMP CSECT
NTIDSAMP AMODE 31
NTIDSAMP RMODE ANY

BAKR R14,0 Save calling programs
LR R12,R15 establishes base reg
USING NTIDSAMP,R12

***********************************************************************
* INITIALIZE THE NAME AND TOKEN FIELDS *
***********************************************************************

MVC NAME,=CL16'NTIDSAMP NAME' INITIALIZE NAME FIELD
MVC TOKEN,NAME FOR EXAMPLE, MAKE TOKEN THE SAME AS THE NAME

***********************************************************************
* TASK LEVEL CREATE EXAMPLE *
***********************************************************************

CALL IEANTCR,(LEVEL,NAME,TOKEN,PERSOPT,RETCODE)

CLC RETCODE,=F'0' IS RETURN CODE 0?
BNE ABEND NO, GO ABEND
EJECT

***********************************************************************
* TASK LEVEL RETRIEVE EXAMPLE *
***********************************************************************

CALL IEANTRT,(LEVEL,NAME,TOKEN,RETCODE)

CLC RETCODE,=F'0' IS RETURN CODE 0?
BNE ABEND NO, GO ABEND
EJECT

***********************************************************************
* TASK LEVEL DELETE EXAMPLE *
***********************************************************************
```
IEANTCR callable service

CALL IEANTDL,(LEVEL,NAME,RETCODE)

***********************************************************************
  CLC  RETCODE,=F'0'   IS RETURN CODE 0?
  BNE  ABEND          NO, GO ABEND
  EJECT
  SLR  R15,R15       SET RETURN CODE OF ZERO
  EXIT  PR            RETURN TO CALLER
  ABEND  ABEND X'BAD' ABEND IF NONZERO RETURN CODE
  EJECT
***********************************************************************

* NAME/TOKEN VARIABLE DECLARES
***********************************************************************
  IEANTASM
  EJECT
***********************************************************************

* Constants and data areas
***********************************************************************

  LEVEL  DC  A(IEANT_TASK_LEVEL)  Task level
  NAME   DS  CL16                 Name for name/token pair
  TOKEN  DS  XL16                 Token for name/token pair
  PERSOPT  DC  A(IEANT_NOPERSIST)  Persist option
  RETCODE DS  F                  Return code

***********************************************************************

* EQUATES
***********************************************************************

R1  EQU  1
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
END NTIDSAMP
Chapter 39. IEANTDL — Delete a name/token pair

Description

Call the IEANTDL service to delete a name/token pair.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state with any PSW key</td>
</tr>
<tr>
<td></td>
<td>Note: Problem-state programs with PSW key 8 - 15 cannot delete:</td>
</tr>
<tr>
<td></td>
<td>• System-level pairs</td>
</tr>
<tr>
<td></td>
<td>• Name/token pairs created by supervisor-state or PSW key 0-7 programs.</td>
</tr>
</tbody>
</table>

| Dispatchable unit mode | Task or SRB |
|                       | Note: SRB-mode callers cannot delete a task-level pair. |
| Cross memory mode     | Any PASN, any HASN, any SASN |
| AMODE                 | 31-bit |
| ASC mode              | Primary or access register (AR) |
| Interrupt status      | Enabled for I/O and external interrupts |
| Locks                 | No locks held |
| Control parameters    | The parameter list and all parameters must reside in the caller's primary address space. |

Programming requirements

Before you use name/token services, you can optionally include the IEANTASM macro to invoke name/token services equate (EQU) statements. IEANTASM provides the following constants for use in your program:

* Name/Token Level Constants

* IEANT_TASK_LEVEL EQU 1
  IEANT_HOME_LEVEL EQU 2
  IEANT_PRIMARY_LEVEL EQU 3
  IEANT_SYSTEM_LEVEL EQU 4
  IEANT_TASKAUTH_LEVEL EQU 11
  IEANT_HOMEAUTH_LEVEL EQU 12
  IEANT_PRIMARYAUTH_LEVEL EQU 13

* Name/Token Persistence Constants

* IEANT_NOPERSIST EQU 0
  IEANT_PERSIST EQU 1

* Name/Token Return Code Constants

* IEANT_OK EQU 0
  IEANT_DUP_NAME EQU 4
  IEANT_NOT_FOUND EQU 4
  IEANT_24BITMODE EQU 8
  IEANT_NOT_AUTH EQU 16
  IEANT_SRБ_MODE EQU 20
  IEANT_LOCK_HELD EQU 24
  IEANT_LEVEL_INVALID EQU 28

© Copyright IBM Corp. 1988, 2015
IEANTDL callable service

IEANT_NAME_INVALID EQU 32
IEANT_PERSIST_INVALID EQU 36
IEANT_AR_INVALID EQU 40
IEANT_UNEXPECTED_ERR EQU 64

Restrictions
Do not use the IEANTDL callable service in a RESMGR resource manager routine unless one of the following is true:
- The name/token pair is a system-level persistent name/token pair.
- The resource manager is running for a daughter task of the task that owns the name/token pair.
- The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.

Input register information
Before issuing the IEANTDL callable service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.
IEANTDL callable service

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEANTDL</td>
<td>(level, user_name, return_code)</td>
</tr>
</tbody>
</table>

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEANTDL:

1. LOAD EP=IEANTDL
   Save the entry point address
   (...) Put the saved entry point address into R15
   CALL (15), (...) 

2. L 15,X'10'
   L 15,X'220'(15,0)
   L 15,X'14'(15,0)
   L 15,X'0C'(15,0)
   CALL (15), (...) 

This second technique requires AMODE=31, and, before the CALL is issued, verification that the IEANTDL service is supported by the system (in the CVT, both the CVTOSEXT and the CVTOS390 bits are set on).

Parameters

The parameters are explained as follows:

level

Specifies a fullword that contains an integer indicating the level of the name/token pair you wish to delete:

1 Task
2 Home address space
3 Primary address space
4 System.

user_name

Specifies the 16-byte area containing the name of the name/token pair to be deleted.

return_code

Specifies a fullword to contain the return code from the IEANTDL service.

ABEND codes

The caller might encounter abend X'AC7' with a reason code of either X'00030000' or X'00030001'. See z/OS MVS System Codes for an explanation and responses for these codes.

Return and reason codes

When IEANTDL returns control to your program, GPR 15 and return_code contain a return code. The following table identifies return codes in hexadecimal, tells what each means, and recommends an action that you need to take.
### IEANTDL callable service

**Table 46. Return Codes for the IEANTDL Macro**

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | **Meaning**: The operation was successful.  
|                         | **Action**: None. |
| 04                      | **Meaning**: The request is rejected because the system could not find the requested name/token pair.  
|                         | **Action**: Check the user_name you specified. |
| 08                      | **Meaning**: The request is rejected because the caller is in 24-bit addressing mode.  
|                         | **Action**: Change your program to 31-bit addressing mode. |
| 10                      | **Meaning**: An unauthorized caller attempted to delete a system-level pair or a name/token pair that was created by an authorized program.  
|                         | **Action**: Check which level of name/token pair you are deleting. |
| 14                      | **Meaning**: An SRB-mode caller attempted to delete a task-level name/token pair.  
|                         | **Action**: Change the program to task mode or check the value you set for the level parameter. |
| 18                      | **Meaning**: The caller held locks.  
|                         | **Action**: Release all locks before issuing IEANTDL. |
| 1C                      | **Meaning**: The caller specified an incorrect level.  
|                         | **Action**: Respecify the correct level. Valid options are 1, 2, 3, or 4. |
| 20                      | **Meaning**: The caller specified an incorrect user_name.  
|                         | **Action**: Respecify the correct user_name. |
| 28                      | **Meaning**: The caller was in AR ASC mode and AR1 was not zero.  
|                         | **Action**: Change your program to primary mode or set AR1 to zero. |
| 40                      | **Meaning**: A system error occurred while handling the request.  
|                         | **Action**: Retry the request. |

### Example

For a complete example of creating, retrieving, and deleting a task-level name/token pair, see the IEANTCR callable service.
Chapter 40. IEANTRT — Retrieve the token from a name/token pair

Description

Call the IEANTRT service to retrieve the token from a name/token pair.

The IEANTRT callable service can also be used to obtain the name of the logrec medium, either the name of the logrec data set or the name of the logrec log stream.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state, with any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>The caller can hold a local, CML, or CMS lock; however, no locks are required.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>The parameter list and all parameters must reside in the caller's primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

Before you use name/token services, you can optionally include macro IEANTASM to invoke name/token services equate (EQU) statements. IEANTASM provides the following constants for use in your program:

* Name/Token Level Constants

* IEANT_TASK_LEVEL EQU 1
* IEANT_HOME_LEVEL EQU 2
* IEANT_PRIMARY_LEVEL EQU 3
* IEANT_SYSTEM_LEVEL EQU 4
* IEANT_TASKAUTH_LEVEL EQU 11
* IEANT_HOMEAUTH_LEVEL EQU 12
* IEANT_PRIMARYAUTH_LEVEL EQU 13

* Name/Token Persistence Constants

* IEANT_NOPERSIST EQU 0
* IEANT_PERSIST EQU 1

* Name/Token Return Code Constants

* IEANT_OK EQU 0
* IEANT_DUP_NAME EQU 4
* IEANT_NOT_FOUND EQU 4
* IEANT_24BITMODE EQU 8
* IEANT_NOT_AUTH EQU 16
* IEANT_SRBMODE EQU 20
* IEANT_LOCK_HELD EQU 24
IEANTRT callable service

IEANT_LEVEL_INVALID EQU 28
IEANT_NAME_INVALID EQU 32
IEANT_PERSIST_INVALID EQU 36
IEANT_AR_INVALID EQU 40
IEANT_UNEXPECTED_ERR EQU 64

To obtain the name of the logrec data set or the name of the logrec log stream, you can include the IFBNTASM macro, as well as the IEANTASM macro, in your program. See “Example 2” on page 536 for the list of definitions IFBNTASM provides.

Restrictions

- Do not use the IEANTRT callable service in a RESMGR resource manager routine unless one of the following is true:
  - The name/token pair is a system-level persistent name/token pair.
  - The resource manager is running for a daughter task of the task that owns the name/token pair.
  - The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.
- Do not call the IEANTRT callable service with user_name and user_token parameters being the same storage locations.

Input register information

Before issuing the IEANTRT callable service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register  
Contents
0-1  Used as work registers by the system
2-13  Unchanged
14  Used as a work register by the system
15  Return code

When control returns to the caller, the access registers (ARs) contain:

Register  
Contents
0-1  Used as work registers by the system
2-13  Unchanged
14-15  Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.
IEANTRT callable service

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEANTRT</td>
<td>,(level, user_name, user_token, return_code)</td>
</tr>
</tbody>
</table>

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEANTRT:

1. LOAD EP=IEANTRT
   Save the entry point address
   (...) Put the saved entry point address into R15
   CALL R15, (...)  

2. L 15,X'10'
   L 15,X'220'(15,0)
   L 15,X'14'(15,0)
   L 15,X'08'(15,0)
   CALL R15, (...)  

This second technique requires AMODE=31, and, before the CALL is issued, verification that the IEANTRT service is supported by the system (in the CVT, both the CVTOSEXT and the CVTOS390 bits are set on).

Parameters
The parameters are explained as follows:

**level**
Specifies a fullword that contains an integer indicating the level of the name/token pair from which you want to retrieve the token:

- 1 Task
- 2 Home address space
- 3 Primary address space
- 4 System
- 11 Task with authorization check
- 12 Home address space with authorization check
- 13 Primary address space with authorization check.

**Note:** Levels 11, 12, and 13 indicate that the IEANTRT callable service should determine if the name/token pair being retrieved was created by an authorized program.
IEANTRT callable service

\texttt{\textbf{user\_name}}
\begin{quote}
Specifies the 16-byte area containing the name of the requested name/token pair.
\end{quote}
\texttt{\textbf{user\_token}}
\begin{quote}
Specifies the 16-byte area to contain the token of the requested name/token pair.
\end{quote}
\texttt{\textbf{return\_code}}
\begin{quote}
Specifies a fullword to contain the return code from the IEANTRT service.
\end{quote}

\textbf{ABEND codes}
None.

\textbf{Return codes}
When IEANTRT returns control to your program, GPR 15 and \textit{return\_code} contain a return code. The following table identifies return codes in hexadecimal, tells what each means, and recommends as action that you need to take.

\begin{table}[h]
\centering
\caption{Return Codes for the IEANTRT Macro}
\begin{tabular}{|c|p{\textwidth}|}
\hline
\textbf{Hexadecimal Return Code} & \textbf{Meaning and Action} \\
\hline
00 & \textbf{Meaning}: The operation was successful. \\
 & \textbf{Action}: None. \\
\hline
04 & \textbf{Meaning}: The request is rejected because the system could not find the requested name/token pair. \\
 & \textbf{Action}: Check the \textit{user\_name} you specified. \\
\hline
08 & \textbf{Meaning}: The request is rejected because the caller is in 24-bit addressing mode. \\
 & \textbf{Action}: Change your program to 31-bit addressing mode. \\
\hline
10 & \textbf{Meaning}: A request for a retrieval with authorization check attempted to retrieve a name/token pair created by an unauthorized caller. \\
 & \textbf{Action}: If your program is authorized, you need to make sure that the name/token pair you are retrieving was created by another authorized program. You may choose to use the name/token pair if it was created by an unauthorized program, but doing so might cause data integrity problems. \\
\hline
14 & \textbf{Meaning}: An SRB-mode caller attempted to retrieve a task-level name/token pair. \\
 & \textbf{Action}: Check which level of name/token pair you are retrieving. \\
\hline
1C & \textbf{Meaning}: The caller specified an incorrect \textit{level}. \\
 & \textbf{Action}: Respecify the correct \textit{level}. Valid options are 1, 2, 3, 4, 11, 12, or 13. \\
\hline
40 & \textbf{Meaning}: A system error occurred while handling the request. \\
 & \textbf{Action}: Retry the request. \\
\hline
\end{tabular}
\end{table}

\textbf{Example 1}
For a complete example of creating, retrieving, and deleting a task-level name/token pair, see the IEANTRT callable service.

\textbf{Example 2}
Following is an example of using Name/Token services to obtain the name of the logrec data set or logrec log stream. (Note that because the routine is not reentrant,
module IEANTRT is first loaded and then called.) IEANTRT returns a token that contains a pointer to the name of the logrec data set or logrec log stream.

Before you use name/token services, you can optionally include macro IFBNTASM which provides the following definitions for use in your program:

* **IFBNTASM Parameters**
* **IFBNT_DSNLOGREC** DC CL16‘DSNLOGREC’ System level DSNLOGREC name
  
  * IFBNT_VERSION1 EQU X’01’ First version of IFBNT_TOKEN
  * IFBNT_VERSION2 EQU X’02’ Second version of IFBNT_TOKEN
  * IFBNT_LATEST_VERSION EQU X’02’ Latest version of IFBNT_TOKEN

* **IFBNT_TOKEN** DSECT , Token area
* **IFBNT_LOGREC_NAME_PTR** DS A Address of the LOGREC data set name area
* **IFBNT_VERSION** DS X Version of IFBNT_LOGREC
* **IFBNT_RESV1** DS X Reserved for IBM
* **IFBNT_LENGTH** DS XL2 Length of IFBNT_LOGREC area
* **IFBNT_RESV2** DS CL8 Reserved for IBM

* **IFBNT_LOGREC** DSECT , Pointed to by
* **IFBNT_LOGREC_NAME** DS CL44 LOGREC data set name or no data set name string (see comments at end of mapping)
* **IFBNT_LOGREC_CURRENT** DS XL1 Current Logrec recording medium
* **IFBNT_LOGREC_PREVIOUS** DS XL1 Previous Logrec recording medium
* **IFBNT_LOGREC_LOGSTREAM** DS CL26 Logrec log stream name, only filled in when IFBNT_USE_LOGSTREAM is the current medium
* **IFBNT_LOGREC_LEN** EQU *-IFBNT_LOGREC Length of IFBNT_LOGREC

* The following values are used in the following fields:
  * IFBNT_LOGREC_CURRENT
  * IFBNT_LOGREC_PREVIOUS

* **IFBNT_USE_DATASET** EQU X’01’ Logrec data set being used
* **IFBNT_USE_LOGSTREAM** EQU X’02’ Logrec log stream being used
* **IFBNT_IGNORE_RECORDS** EQU X’03’ Logrec recording is ignored

* If a Logrec data set was not defined during the IPL of the system then the following string will appear in field
  * IFBNT_LOGREC_NAME = ‘...NO.LOGREC.DATA.SET.DEFINED...’

**IFBNT_TOKEN** provides a DSECT to map the returned token area.

**IFBNT_LOGREC_NAME_PTR** contains the address of the logrec data set name.

**IFBNT_LOGREC** provides a DSECT to map the logrec recording medium.

**IFBNT_LOGREC_NAME** contains the name of the installation-defined logrec data set or no data set name, if the recording medium is other than a data set.

---

**Title**

"DSNLOGREC Name/Token Retrieve Example Routine"

**IFBNTXMP** AMODE 31

**IFBNTXMP** RMODE ANY

**IFBNTXMP CSECT**

**BAKR** R14,0 Save calling program's
IEANTRT callable service

* registers and return location
  LR R12,R15 Establish base ref
  USING IFBNTXMP,R12 Set addressability
  MODID BRANCH=YES

*********************************************************************
* Initialize the NAME field
*********************************************************************
  MVC NAME,IFBNT_DSNLOGREC Request DSNLOGREC name

*********************************************************************
* System level DSNLOGREC Retrieve example
*********************************************************************
  LOAD EP=IEANTRT Get address of IEANTRT routine
  LR R15,R0 Set address for Call
  CALL (15),(LEVEL,NAME,TOKEN,RETCODE)

  * LA R15,IEANT_OK Get successful return code value
  C R15,RETCODE Was TOKEN Returned?
  BNE ABEND No, Go ABEND
  EJECT

*********************************************************************
* Get the installation specified LOGREC data set name
*********************************************************************
  LA R2,TOKEN Set pointer to TOKEN area
  USING IFBNT_TOKEN,R2 Set addressability
  * DSNLOGREC TOKEN area
  L R2,IFBNT_LOGREC_NAME_PTR Get pointer to data set name
  DROP R2 Free up register 2
  USING IFBNT_LOGREC,R2 Set addressability to
  * LOGREC data set name area

*********************************************************************
* If you are interested in obtaining the log stream name, reference
* IFBNT_LOGREC_LOGSTREAM instead of IFBNT_LOGREC_NAME here,
* using the MVC command to move the log stream name to your
* own program's area.
*********************************************************************
  MVC LOGRNAME,IFBNT_LOGREC_NAME Move LOGREC data set name
  * to own area
  DROP R2 Free up register 2
  EXIT DS 0H Return point
  SLR R15,R15 Set return code of zero
  PR Return to caller
  EJECT
  ABEND ABEND X'BAD' ABEND if non-zero return code
  EJECT

*********************************************************************
* Local working storage declares
*********************************************************************
  NAME DS CL16 Name for Name/Token pair
  TOKEN DS XL16 Token for Name/Token Pair
  RETCODE DS F Return code from IEANTRT
  LOGRNAME DS CL44 Area for LOGREC data set name

*********************************************************************
* Constant and Equates
*********************************************************************
  LEVEL DC A(IEANT_SYSTEM_LEVEL) SYSTEM LEVEL
  R0 EQU 0
  R1 EQU 1
  R2 EQU 2
  R11 EQU 11
  R12 EQU 12
  R13 EQU 13
  R14 EQU 14
  R15 EQU 15
  EJECT

*********************************************************************
* NAME/TOKEN SYSTEM LEVEL DSNLOGREC VARIABLE DECLARES
*********************************************************************
IEANTRT callable service

*********************************************************************
IFBNTASM
EJECT
*********************************************************************
* NAME/TOKEN VARIABLE DECLARES
*********************************************************************
IEANTASM
END IFBNTXMP
IEANTRT callable service
Chapter 41. IEAN4CR — Create a name/token pair

Description

Call the IEAN4CR service to create a name/token pair.

Environment

The requirements for the caller are:

- **Environmental factor**
  - **Requirement**: Problem state, with any PSW key.
  - **Note**: Problem-state programs with PSW key 8 - 15 cannot create system-level pairs.

- **Dispatchable unit mode**
  - **Task or SRB**: Task or SRB callers cannot create a task-level pair.
  - **Note**: SRB-mode callers cannot create a task-level pair.

- **Cross memory mode**: Any PASN, any HASN, any SASN

- **AMODE**: 64-bit

- **ASC mode**: Primary or access register (AR)

- **Interrupt status**: Enabled for I/O and external interrupts

- **Locks**: No locks held

- **Control parameters**: The parameter list and all parameters must reside in the caller's primary address space.

Programming requirements

Before you use name/token services, you can optionally include the IEANTASM macro to invoke name/token services equate (EQU) statements. IEANTASM provides the following constants for use in your program:

* **Name/Token Level Constants**
  - IEANT_TASK_LEVEL EQU 1
  - IEANT_HOME_LEVEL EQU 2
  - IEANT_PRIMARY_LEVEL EQU 3
  - IEANT_SYSTEM_LEVEL EQU 4
  - IEANT_TASKAUTH_LEVEL EQU 11
  - IEANT_HOMEAUTH_LEVEL EQU 12
  - IEANT_PRIMARYAUTH_LEVEL EQU 13

* **Name/Token Persistence Constants**
  - IEANT_NOPERSIST EQU 0
  - IEANT_PERSIST EQU 1
  - IEANT_NOCHECKPOINT EQU 0
  - IEANT_CHECKPOINTOK EQU 2

* **Name/Token Return Code Constants**
  - IEANT_OK EQU 0
  - IEANT_DUP_NAME EQU 4
  - IEANT_NOT_FOUND EQU 4
  - IEANT_24BITMODE EQU 8
  - IEANT_NOT_AUTH EQU 16
  - IEANT_SRB_MODE EQU 20
  - IEANT_LOCK_HELD EQU 24
  - IEANT_LEVEL_INVALID EQU 28
IEAN4CR callable service

Restrictions
Do not use the IEAN4CR callable service in a RESMGR resource manager routine unless one of the following is true:
- The name/token pair is a system-level persistent name/token pair.
- The resource manager is running for a daughter task of the task that owns the name/token pair.
- The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.

Input register information
Before issuing the IEAN4CR callable service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.
IEAN4CR callable service

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTATE AMODE64=YES</td>
<td></td>
</tr>
<tr>
<td>CALL IEAN4CR</td>
<td>,(level ,user_name ,user_token ,persist_option ,return_code)</td>
</tr>
</tbody>
</table>

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEAN4CR:

1. LOAD EP=IEAN4CR
   Save the 8-byte entry point address with bit 63 changed to 0
   (...) 
   Put the saved entry point address with bit 63 changed to 0 into 64-bit R15
   CALL (15),(...) 

2. LLGT 15,X'10'
   L 15,X'220'(15,0)
   L 15,X'14'(15,0)
   L 15,X'7C'(15,0)
   CALL (15),(...) 

Both of these alternate techniques require verification that the IEAN4CR service is available (in the CVT, bit CVTZOS_V1R11 is on indicating that the program is running on z/OS V1R11 or a later release).

Parameters

The parameters are explained as follows:

level
   Specifies a fullword that contains an integer indicating the level of the name/token pair:
   1   Task
   2   Home address space
   3   Primary address space
   4   System.

user_name
   Specifies the 16-byte area containing the name of the name/token pair that the user creates. The bytes of the name may have any value. The name may contain blanks, integers, or addresses.

Names must be unique within a level. Here are some examples.

- Two task-level name/token pairs owned by the same task cannot have the same name. However, two task-level name/token pairs owned by different tasks can have the same name.
- Two home-address-space-level name/token pairs in the same address space cannot have the same name. However, two home-address-space-level name/token pairs in different address spaces can have the same name.

Because of these unique requirements you must avoid using the same names that IBM uses for name/token pairs. Do not use the following names:
**IEAN4CR callable service**

- Names that begin with A through I
- Names that begin with X'00'.

**user_token**
- Specifies the 16-byte area containing the token of the name/token pair that the user creates.

**persist_option**
- Specifies a fullword that contains an integer indicating if a system-level name/token pair should persist after the creating address space's job step task terminates or if Checkpoint/Restart can be issued if the program has this task-level name/token pair. If a program has non-task-level name/token pairs or has task-level name/token pairs that did not specify IEANT_CHECKPOINTOK, the program cannot take a checkpoint.
  - 0 - system-level pair will not persist and checkpoint is not permitted.
  - 1 - system-level pair will persist.
  - 2 - checkpoint is permitted.

Note: Only system-level name/token pairs can persist after the creating task terminates. Only task-level name/token pairs can permit checkpoint. You must specify 0 for all other levels.

**return_code**
- Specifies a fullword to contain the return code from the IEAN4CR service.

**ABEND codes**
- The caller might encounter abend X'AC7' with a reason code of either X'00030000' or X'00030001'. See [z/OS MVS System Codes](z/OS MVS System Codes) for an explanation and responses for these codes.

**Return and reason codes**
- When IEAN4CR returns control to your program, GPR 15 and return_code contain a return code. The following table identifies return codes in hexadecimal and decimal, tells what each means, and recommends an action that you need to take:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | Meaning: The operation was successful.  
Action: None.            |
| 04                      | Meaning: The user_name specified already exists.  
Action: Choose a different user_name. |
| 08                      | Meaning: The request is rejected because the caller is in 24-bit addressing mode.  
Action: Change your program to 64-bit addressing mode. |
| 10                      | Meaning: An unauthorized caller attempted to create a system-level name/token pair.  
Action: Check which level of name/token pair you are creating. |
| 14                      | Meaning: An SRB-mode caller attempted to create a task-level name/token pair.  
Action: Change your program to task mode or use a different level. |
| 18                      | Meaning: The caller held locks.  
Action: Release all locks before issuing IEAN4CR. |
### Return Codes for the IEAN4CR Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| **1C**                  | **Meaning:** The caller specified an incorrect level.  
                          | **Action:** Respecify the correct level. Valid options are 1, 2, 3, or 4. |
| **20**                  | **Meaning:** The caller specified an incorrect user_name.  
                          | **Action:** Respecify the correct user_name. |
| **24**                  | **Meaning:** The caller specified an incorrect persist_option.  
                          | **Action:**  
                          | • For system-level name/token pairs, you must specify zero or one for the persist_option.  
                          | • For task-level name/token pairs, you must specify zero or two for the persist_option.  
                          | • For home or primary address space level name/token pairs, you must specify zero for the persist_option. |
| **28**                  | **Meaning:** The caller was in AR ASC mode and AR1 was not zero.  
                          | **Action:** Change your program to primary mode or set AR1 to zero. |
| **40**                  | **Meaning:** A system error occurred while handling the request.  
                          | **Action:** Retry the request. |
IEAN4CR callable service
Chapter 42. IEAN4DL — Delete a name/token pair

Description

Call the IEAN4DL service to delete a name/token pair.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state with any PSW key</td>
</tr>
<tr>
<td>Note:</td>
<td>Problem-state programs with PSW key 8 - 15 cannot delete:</td>
</tr>
<tr>
<td></td>
<td>• System-level pairs</td>
</tr>
<tr>
<td></td>
<td>• Name/token pairs created by supervisor-state or PSW key 0-7 programs.</td>
</tr>
</tbody>
</table>

| Dispatchable unit mode | Task or SRB |
| Note:                 | SRB-mode callers cannot delete a task-level pair. |

| Cross memory mode          | Any PASN, any HASN, any SASN |
| AMODE                      | 64-bit |
| ASC mode                   | Primary or access register (AR) |
| Interrupt status           | Enabled for I/O and external interrupts |
| Locks                      | No locks held |
| Control parameters         | The parameter list and all parameters must reside in the caller's primary address space. |

Programming requirements

Before you use name/token services, you can optionally include the IEANTASM macro to invoke name/token services equate (EQU) statements. IEANTASM provides the following constants for use in your program:

* Name/Token Level Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEANT_TASK_LEVEL</td>
<td>1</td>
</tr>
<tr>
<td>IEANT_HOME_LEVEL</td>
<td>2</td>
</tr>
<tr>
<td>IEANT_PRIMARY_LEVEL</td>
<td>3</td>
</tr>
<tr>
<td>IEANT_SYSTEM_LEVEL</td>
<td>4</td>
</tr>
<tr>
<td>IEANT_TASKAUTH_LEVEL</td>
<td>11</td>
</tr>
<tr>
<td>IEANT_HOMEAUTH_LEVEL</td>
<td>12</td>
</tr>
<tr>
<td>IEANT_PRIMARYAUTH_LEVEL</td>
<td>13</td>
</tr>
</tbody>
</table>

* Name/Token Persistence Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEANT_NOPERSIST</td>
<td>0</td>
</tr>
<tr>
<td>IEANT_PERSIST</td>
<td>1</td>
</tr>
</tbody>
</table>

* Name/Token Return Code Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEANT_OK</td>
<td>0</td>
</tr>
<tr>
<td>IEANT_DUP_NAME</td>
<td>4</td>
</tr>
<tr>
<td>IEANT_NOT_FOUND</td>
<td>4</td>
</tr>
<tr>
<td>IEANT_24BITMODE</td>
<td>8</td>
</tr>
<tr>
<td>IEANT_NOT_AUTH</td>
<td>16</td>
</tr>
<tr>
<td>IEANT_SRB_MODE</td>
<td>20</td>
</tr>
<tr>
<td>IEANT_LOCK_HELD</td>
<td>24</td>
</tr>
<tr>
<td>IEANT_LEVEL_INVALID</td>
<td>28</td>
</tr>
</tbody>
</table>
IEAN4DL callable service

IEANT_NAME_INVALID  EQU 32
IEANT_PERSIST_INVALID EQU 36
IEANT_AR_INVALID    EQU 40
IEANT_UNEXPECTED_ERR EQU 64

Restrictions
Do not use the IEAN4DL callable service in a RESMGR resource manager routine unless one of the following is true:

- The name/token pair is a system-level persistent name/token pair.
- The resource manager is running for a daughter task of the task that owns the name/token pair.
- The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.

Input register information
Before issuing the IEAN4DL callable service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.
IEAN4DL callable service

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTATE AMODE64=YES</td>
<td></td>
</tr>
<tr>
<td>CALL IEAN4DL</td>
<td>,(level</td>
</tr>
<tr>
<td></td>
<td>,user_name</td>
</tr>
<tr>
<td></td>
<td>,return_code)</td>
</tr>
</tbody>
</table>

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEAN4DL:

1. LOAD EP=IEAN4DL
   - Save the 8-byte entry point address with bit 63 changed to 0
   - Put the saved entry point address with bit 63 changed to 0 into 64-bit R15
   - CALL (15),(...)

2. LLGT 15,X'10'
   - L 15,X'220'(15,0)
   - L 15,X'14'(15,0)
   - L 15,X'84'(15,0)
   - CALL (15),(...)

Both of these alternate techniques require verification that the IEAN4DL service is available (in the CVT, bit CVTZOS_V1R11 is on indicating that the program is running on z/OS V1R11 or a later release).

Parameters

The parameters are explained as follows:

1. **level**
   - Specifies a fullword that contains an integer indicating the level of the name/token pair you wish to delete:
     - 1 Task
     - 2 Home address space
     - 3 Primary address space
     - 4 System.

2. **user_name**
   - Specifies the 16-byte area containing the name of the name/token pair to be deleted.

3. **return_code**
   - Specifies a fullword to contain the return code from the IEAN4DL service.

ABEND codes

The caller might encounter abend X'AC7' with a reason code of either X'00030000' or X'00030001'. See [z/OS MVS System Codes](#) for an explanation and responses for these codes.

Return and reason codes

When IEAN4DL returns control to your program, GPR 15 and return_code contain a return code. The following table identifies return codes in hexadecimal, tells what each means, and recommends an action that you need to take.
### IEAN4DL callable service

**Table 49. Return Codes for the IEAN4DL Macro**

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | **Meaning**: The operation was successful.  
**Action**: None. |
| 04                      | **Meaning**: The request is rejected because the system could not find the requested name/token pair.  
**Action**: Check the *user_name* you specified. |
| 08                      | **Meaning**: The request is rejected because the caller is in 24-bit addressing mode.  
**Action**: Change your program to 64-bit addressing mode. |
| 10                      | **Meaning**: An unauthorized caller attempted to delete a system-level pair or a name/token pair that was created by an authorized program.  
**Action**: Check which level of name/token pair you are deleting. |
| 14                      | **Meaning**: An SRB-mode caller attempted to delete a task-level name/token pair.  
**Action**: Change the program to task mode or check the value you set for the level parameter. |
| 18                      | **Meaning**: The caller held locks.  
**Action**: Release all locks before issuing IEAN4DL. |
| 1C                      | **Meaning**: The caller specified an incorrect *level*.  
**Action**: Respecify the correct *level*. Valid options are 1, 2, 3, or 4. |
| 20                      | **Meaning**: The caller specified an incorrect *user_name*.  
**Action**: Respecify the correct *user_name*. |
| 28                      | **Meaning**: The caller was in AR ASC mode and AR1 was not zero.  
**Action**: Change your program to primary mode or set AR1 to zero. |
| 40                      | **Meaning**: A system error occurred while handling the request.  
**Action**: Retry the request. |
Chapter 43. IEAN4RT — Retrieve the token from a name/token pair

Description

Call the IEAN4RT service to retrieve the token from a name/token pair.

The IEAN4RT callable service can also be used to obtain the name of the logrec medium, either the name of the logrec data set or the name of the logrec log stream.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state, with any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Note: SRB-mode callers cannot retrieve a task-level pair.</td>
<td></td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller can hold a local, CML, or CMS lock; however, no locks are required.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>The parameter list and all parameters must reside in the caller's primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

Before you use name/token services, you can optionally include macro IEANTASM to invoke name/token services equate (EQU) statements. IEANTASM provides the following constants for use in your program:

* Name/Token Level Constants

<table>
<thead>
<tr>
<th>Name/Token Level Constant</th>
<th>EQU Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEANT_TASK_LEVEL</td>
<td>1</td>
</tr>
<tr>
<td>IEANT_HOME_LEVEL</td>
<td>2</td>
</tr>
<tr>
<td>IEANT_PRIMARY_LEVEL</td>
<td>3</td>
</tr>
<tr>
<td>IEANT_SYSTEM_LEVEL</td>
<td>4</td>
</tr>
<tr>
<td>IEANT_TASKAUTH_LEVEL</td>
<td>11</td>
</tr>
<tr>
<td>IEANT_HOMEAUTH_LEVEL</td>
<td>12</td>
</tr>
<tr>
<td>IEANT_PRIMARYAUTH_LEVEL</td>
<td>13</td>
</tr>
</tbody>
</table>

* Name/Token Persistence Constants

<table>
<thead>
<tr>
<th>Name/Token Persistence Constant</th>
<th>EQU Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEANT_NOPERSIST</td>
<td>0</td>
</tr>
<tr>
<td>IEANT_PERSIST</td>
<td>1</td>
</tr>
</tbody>
</table>

* Name/Token Return Code Constants

<table>
<thead>
<tr>
<th>Name/Token Return Code Constant</th>
<th>EQU Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEANT_OK</td>
<td>0</td>
</tr>
<tr>
<td>IEANT_DUP_NAME</td>
<td>4</td>
</tr>
<tr>
<td>IEANT_NOT_FOUND</td>
<td>4</td>
</tr>
<tr>
<td>IEANT_24BITMODE</td>
<td>8</td>
</tr>
<tr>
<td>IEANT_NOT_AUTH</td>
<td>16</td>
</tr>
<tr>
<td>IEANT_SRB_MODE</td>
<td>20</td>
</tr>
<tr>
<td>IEANT_LOCK_HELD</td>
<td>24</td>
</tr>
</tbody>
</table>
IEAN4RT callable service

IEANT_LEVEL_INVALID EQU 28
IEANT_NAME_INVALID EQU 32
IEANT_PERSIST_INVALID EQU 36
IEANT_AR_INVALID EQU 40
IEANT_UNEXPECTED_ERR EQU 64

To obtain the name of the logrec data set or the name of the logrec log stream, you can include the IFBNTASM macro, as well as the IEANTASM macro, in your program. See "Example 2" on page 536 for the list of definitions IFBNTASM provides.

Restrictions

- Do not use the IEAN4RT callable service in a RESMGR resource manager routine unless one of the following is true:
  - The name/token pair is a system-level persistent name/token pair.
  - The resource manager is running for a daughter task of the task that owns the name/token pair.
  - The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.
- Do not call the IEAN4RT callable service with user_name and user_token parameters being the same storage locations.

Input register information

Before issuing the IEAN4RT callable service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.
Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTATE AMODE64=YES</td>
<td></td>
</tr>
<tr>
<td>CALL IEAN4RT</td>
<td>(level, user_name, user_token, return_code)</td>
</tr>
</tbody>
</table>

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEAN4RT:

1. LOAD EP=IEAN4RT
   Save the 8-byte entry point address with bit 63 changed to 0
   (...)
   Put the saved entry point address with bit 63 changed to 0 into 64-bit R15
   CALL (15),(...)

2. LLGT 15,X'10'
   L 15,X'220'(15,0)
   L 15,X'14'(15,0)
   L 15,X'80'(15,0)
   CALL (15),(...)

Both of these alternate techniques require verification that the IEAN4RT service is available (in the CVT, bit CVTZOS_V1R11 is on indicating that the program is running on z/OS V1R11 or a later release).

Parameters
The parameters are explained as follows:

level
Specifies a fullword that contains an integer indicating the level of the name/token pair from which you want to retrieve the token:

1 Task
2 Home address space
3 Primary address space
4 System
11 Task with authorization check
12 Home address space with authorization check
13 Primary address space with authorization check.

Note: Levels 11, 12, and 13 indicate that the IEAN4RT callable service should determine if the name/token pair being retrieved was created by an authorized program.
IEAN4RT callable service

\texttt{\_user_name}

Specifies the 16-byte area containing the name of the requested name/token pair.

\texttt{\_user_token}

Specifies the 16-byte area to contain the token of the requested name/token pair.

\texttt{\_return_code}

Specifies a fullword to contain the return code from the IEAN4RT service.

ABEND codes

None.

Return codes

When IEAN4RT returns control to your program, GPR 15 and \texttt{return\_code} contain a return code. The following table identifies return codes in hexadecimal, tells what each means, and recommends as action that you need to take.

\begin{table}[h]
\caption{Return Codes for the IEAN4RT Macro}
\begin{tabular}{|c|p{10cm}|}
\hline
\textbf{Hexadecimal Return Code} & \textbf{Meaning and Action} \\
\hline
00 & Meaning: The operation was successful.  
\textbf{Action:} None. \\
\hline
04 & Meaning: The request is rejected because the system could not find the requested name/token pair.  
\textbf{Action:} Check the \texttt{user\_name} you specified. \\
\hline
08 & Meaning: The request is rejected because the caller is in 24-bit addressing mode.  
\textbf{Action:} Change your program to 64-bit addressing mode. \\
\hline
10 & Meaning: A request for a retrieval with authorization check attempted to retrieve a name/token pair created by an unauthorized caller.  
\textbf{Action:} If your program is authorized, you need to make sure that the name/token pair you are retrieving was created by another authorized program. You may choose to use the name/token pair if it was created by an unauthorized program, but doing so might cause data integrity problems. \\
\hline
14 & Meaning: An SRB-mode caller attempted to retrieve a task-level name/token pair.  
\textbf{Action:} Check which level of name/token pair you are retrieving. \\
\hline
1C & Meaning: The caller specified an incorrect level.  
\textbf{Action:} Respecify the correct level. Valid options are 1, 2, 3, 4, 11, 12, or 13. \\
\hline
40 & Meaning: A system error occurred while handling the request.  
\textbf{Action:} Retry the request. \\
\hline
\end{tabular}
\end{table}
Chapter 44. IEARBUP — RB update service

Description
IEARBUP allows you to request that the system update the instruction address in the PSW copy in the RB.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state and PSW key 0.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks are required. The caller may hold a local lock, the CMS lock or the CPU lock.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements
The caller must include the CVT and IHAECVT mapping macros.

Restrictions
If the caller holds the CPU lock, the parameter list must be in fixed or DREF storage.

Input register information
Before issuing the IEARBUP macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
IEARBUP RB update service

2-13  Unchanged
14-15  Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
The IEARBUP macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEARBUP.</td>
</tr>
<tr>
<td></td>
<td>IEARBUP</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEARBUP.</td>
</tr>
<tr>
<td>WHICHRB=CURRENT</td>
<td></td>
</tr>
<tr>
<td>WHICHRB=PREV</td>
<td></td>
</tr>
<tr>
<td>WHICHRB=EXPLICIT</td>
<td></td>
</tr>
<tr>
<td>,RB=xrb</td>
<td></td>
</tr>
<tr>
<td>FUNCTION=UPDATE</td>
<td>Default: FUNCTION=UPDATE</td>
</tr>
<tr>
<td>,PSWBYTE03=NO</td>
<td></td>
</tr>
<tr>
<td>,PSWBYTE03=YES</td>
<td></td>
</tr>
<tr>
<td>,ADDRTYPE=NO_CHANGE</td>
<td>Default: ADDRTYPE=NO_CHANGE</td>
</tr>
<tr>
<td>,ADDRTYPE=INRBOPSWA</td>
<td></td>
</tr>
<tr>
<td>,ADDRTYPE=ACTUAL</td>
<td></td>
</tr>
<tr>
<td>,PSWADDR=pswaddr</td>
<td>pswaddr: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>,AMODE=UNCHANGED</td>
<td>Default: AMODE=UNCHANGED</td>
</tr>
<tr>
<td>,AMODE=24</td>
<td></td>
</tr>
<tr>
<td>,AMODE=31</td>
<td></td>
</tr>
<tr>
<td>,AMODE=64</td>
<td></td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRTYPE=DELTA</td>
<td></td>
</tr>
<tr>
<td>PSWDELTA=pswdelta</td>
<td>pswdelta: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>FUNCTION=EXTRACTPSW</td>
<td></td>
</tr>
<tr>
<td>PSWG=xpswg</td>
<td></td>
</tr>
<tr>
<td>RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12)</td>
</tr>
<tr>
<td>RSNCODE=rsncoxe</td>
<td>rsncoxe: RS-type address or register (2) - (12)</td>
</tr>
<tr>
<td>PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>PLISTVER=0</td>
<td></td>
</tr>
<tr>
<td>MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>MF=(L,list addr)</td>
<td>list addr: RS-type address or register (1) - (12)</td>
</tr>
<tr>
<td>MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

- **name**
  - An optional symbol, starting in column 1, that is the name on the IEARBUP macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

- **WHICHRB=**
  - A required parameter that identifies the RB to be updated

- **WHICHRB=CURRENT**
  - indicates to update the current RB.

- **WHICHRB=PREV**
  - indicates to update the previous (older) RB.

- **WHICHRB=EXPLICIT**
  - indicates to update the provided RB. The calling program must ensure that there is proper serialization to keep the provided RB valid for the duration of IEARBUP service processing.

- **RB=xrb**
  - indicates the name (RS-type) or address in register (2)-(12) of a required character input that identifies the RB to be updated.
IEARBUP RB update service

**FUNCTION=** indicates an optional keyword input that identifies the function to be performed

**FUNCTION=UPDATE**
indicates to update an RB

,**PSWBYTE03=**
A required parameter that indicates whether the user has updated the first 4 bytes (bytes 0 to 3) of RBOPSW. If so, the system should use those updated values.

,**PSWBYTE03=NO**
indicates that bytes 0 to 3 were not modified.

,**PSWBYTE03=YES**
indicates that bytes 0 to 3 were modified.

,**ADDRTYPE=**
An optional parameter that identifies the method by which the instruction address in the PSW is provided. The default is ADDRTYPE=NO_CHANGE.

,**ADDRTYPE=NO_CHANGE**
indicates that the instruction address has not been changed.

,**ADDRTYPE=INRBOPSWA**
indicates that the instruction address has been updated in RBOPSWA, along with the one or more AMODE indicators.

,**ADDRTYPE=ACTUAL**
indicates that the instruction address is to be used as is.

,**ADDRTYPE=DELTA**
indicates that the value provided is a delta to the existing address.

,**PSWDELTA=pswdelta**
When ADDRTYPE=DELTA is specified, a required input parameter that contains the delta to be added to the instruction address in the PSW copy stored in the RB. The value is treated as a signed quantity, so a value of X'FFFFFFFE' would be treated as negative two, resulting in subtracting two from the instruction address. The AMODE will remain unchanged.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,**PSWADDR=pswaddr**
When ADDRTYPE=ACTUAL is specified, a required input parameter that contains the address to be placed into the PSW stored in the RB. The high 33 bits must be zero, unless the result is to be AMODE 64.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-byte field.

,**AMODE=**
When ADDRTYPE=ACTUAL is specified, a required parameter that identifies the resulting AMODE for the PSW

,**AMODE=UNCHANGED**
indicates not to change the AMODE.

,**AMODE=24**
indicates to set the AMODE to 24.
IEARBUP RB update service

,.AMODE=31
    indicates to set the AMODE to 31.

,.AMODE=64
    indicates to set the AMODE to 64.

FUNCTION=EXTRACTPSW
    indicates to extract the 128–bit PSW associated with this RB

PSWG=xpswg
    indicates the name (RS-type) or address in register (2)-(12) of the
    required 16–character output that is to contain the 128–bit
    x/Architecture PSW.

    Note: If running under ESA/390 architecture (ARCHLVL 1), the PSW
    is the 128–bit z/Architecture analog of the 64–bit ESA/390 PSW.

,.RETCODE=retcode
    An optional output parameter into which the return code is to be copied from
    GPR 15.

    To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,.RSNCODE=rsncode
    An optional output parameter into which the reason code is to be copied from
    GPR 0.

    To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,.PLISTVER=
    An optional input parameter that specifies the version of the macro. PLISTVER
determines which parameter list the system generates. PLISTVER is an
optional input parameter on all forms of the macro, including the list form.
When using PLISTVER, specify it on all macro forms used for a request and
and with the same value on all of the macro forms. The values are:

,.PLISTVER=IMPLIED_VERSION
    indicates the lowest version that allows all parameters specified on the
    request to be processed. If you omit the PLISTVER parameter,
    IMPLIED_VERSION is the default.

,.PLISTVER=MAX
    indicates the parameter list will be the largest size currently possible. This
    size might grow from release to release and affect the amount of storage
    that your program needs.

    If you can tolerate the size change, IBM recommends that you always
    specify PLISTVER=MAX on the list form of the macro. Specifying MAX
    ensures that the list-form parameter list is always long enough to hold all
    the parameters you might specify on the execute form, when both are
    assembled with the same level of the system. In this way, MAX ensures
    that the parameter list does not overwrite nearby storage.

,.PLISTVER=0
    indicates that you want to use the currently available parameters.

,.MF=
    An optional input parameter that specifies the macro form.

,.MF=S
    Specifies the standard form of the macro, which builds an inline parameter
    list and generates the macro invocation to transfer control to the service.
    MF=S is the default.
IEARBUP RB update service

\texttt{,MF=L,\textit{list addr}}

Specifies the \textit{list} form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

\texttt{,\textit{list addr}}

The name (RS-type) or address in register (1)-(12) of the storage area that contains the parameters.

\texttt{,attr}

An optional 1- to 60-byte input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code \textit{attr}, the system provides a value of 0D.

\texttt{,MF=E,\textit{list addr},COMPLETE}

Specifies the \textit{execute} form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

\texttt{,\textit{list addr}}

The name (RS-type) or address in register (1)-(12) of the storage area that contains the parameters.

\texttt{COMPLETE}

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

**ABEND codes**

The caller may get the following abend code:

\textbf{0C2-02}

The caller was not in supervisor state.

\textbf{0C4-04}

The caller was not in key 0.

**Return and reason codes**

When the IEARBUP macro returns control to your program:

\begin{itemize}
  \item GPR 15 (and \textit{recode}, when you code RETCODE) contains a return code.
  \item When the value in GPR 15 is not zero, GPR 0 (and \textit{rsncode}, when you code RSNCODE) contains a reason code.
\end{itemize}

Macro IEARBUPM provides equate symbols for the return and reason codes.

\begin{tabular}{|l|}
  \hline
  Table 51 on page 561 identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the xxxx value.
  \hline
\end{tabular}
### IEARBUP RB update service

#### Table 51. Return and Reason Codes for the IEARBUP Macro

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol, Meaning, and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>—</td>
<td>Equate Symbol: IearbupRc_OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: Iearbup request successful.</td>
</tr>
<tr>
<td>8</td>
<td>—</td>
<td>Equate Symbol: IearbupRc_InvParm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: Iearbup request specifies invalid parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Refer to the action provided with the specific reason code.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0801</td>
<td>Equate Symbol: IearbupRsnBadVersion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: The version field in the parameter list is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check for possible storage overlay.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0802</td>
<td>Equate Symbol: IearbupRsnBadAMODEField</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: The amode field in the parameter list is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check for possible storage overlay.</td>
</tr>
<tr>
<td>8</td>
<td>xxxx0803</td>
<td>Equate Symbol: IearbupRsnBadAddress</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: The address provided is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Only provide an instruction address that is less than (X'80000000').</td>
</tr>
<tr>
<td>C</td>
<td>—</td>
<td>Equate Symbol: IearbupRc_Env</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: Environmental error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Refer to the action provided with the specific reason code.</td>
</tr>
<tr>
<td>C</td>
<td>xxxx0C01</td>
<td>Equate Symbol: IearbupRsnPrevRBNotFound</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: RB=PREV was requested, but there is only one RB for the current task.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Use RB=CURRENT when there is only one RB.</td>
</tr>
<tr>
<td>C</td>
<td>xxxx0C02</td>
<td>Equate Symbol: IearbupRsnBadAMODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: AMODE=64 was specified but the architecture level is not z/Architecture.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Only request AMODE=64 when the architecture level is z/Architecture.</td>
</tr>
</tbody>
</table>

### Example 1

#### Operation

1. Update the instruction address in the PSW copy stored in the RB to the address provided in field P.

The code is as follows:

```plaintext
IEARBUP ADDRTYPE=ACTUAL,PSWADDR=P,RETCODE=RC,MF=(E,MFL)
...
IEARBUP MF=(L,MFL)
P DS XL8
RC DS F
```
IEARBUP RB update service

Example 2

Operation:
1. Decrement the instruction address in the PSW copy in the RB by 4

The code is as follows:

IEARBUP ADDRTYPE=DELTA,PSWDELTA=PD,RETCODE=RC,MF=(E,MFL)
...
IEARBUP MF=(L,MFL)
PD  DC  F'-4'
RC  DS  F
Chapter 45. IEATDUMP — Transaction dump request

Description

Transaction dump is a service used to request an unformatted dump of virtual storage to a data set, similar to a SYSMDUMP. It is invoked with the IEATDUMP assembler macro, which issues SVC 51. The service is available to both authorized and unauthorized callers; however, not all functions are available to unauthorized callers. If an unauthorized caller requests a transaction dump with authorized keywords, the request will be rejected and message IEA820I will be issued indicating this condition. The transaction dump can be written to one or more automatically allocated data sets by specifying a data set name pattern, similar to the pattern used for the operator DUMPDS NAME=parameter. Automatic allocation reduces the exposure that a dump is truncated because of space constraints, and is done using the generic allocation unit name of SYSALLDA. When a dump is written, messages IEA822I or IEA827I are issued indicating whether the dump is complete or partial.

When a transaction dump is written, a dump directory record describing the dump may be written. The dump directory to be used is specified on the dump request using the IDX keyword. If no dump directory is specified on the request, the directory allocated to IPCSDDIR in the current job step will be used. If no dump directory is specified and IPCSDDIR is not allocated, no record describing the dump will be written.

Dump suppression occurs using symptoms available in the current SDWA or a symptom string may be provided (via the SYMREC keyword). If a symptom string is provided and an SDWA exists, the symptom string is used for suppression purposes. Statistics for dump suppression are contained in the DAE data set and are not differentiated from SYSMDUMPs. If a dump is requested but not taken because it was suppressed, message IEA820I is issued indicating this condition.

Authorized users may specify the REMOTE keyword, on a transaction dump invocation, to request that other address spaces on the current or other MVS images (in the same sysplex) be dumped. When remote dumps are requested, automatic allocation must also be used. Transaction dump uses an incident token to associate this dump with other diagnostic information. Automatic allocation also uses this incident token for symbol substitution in the data set name pattern. An incident token may be generated using the IEAINTKN macro and provided on the dump request using the INTOKEN keyword. If an incident token is not provided, one will be generated and used internally. While an incident token may always be specified, it may be especially important when remote dumps are requested.

Environment

The requirements for the caller are:

Environmental factor          Requirement
Minimum authorization:         Problem state and PSW key 8-15. Use of some keywords is restricted to authorized callers (supervisor state, PSW key 0-7 or APF-authorized).
Dispatchable unit mode:        Task
Cross memory mode:             PASN=HASN=SASN
AMODE:                         24- or 31-bit
IEATDUMP transaction dump

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller must not hold any locks.</td>
</tr>
</tbody>
</table>
| Control parameters:  | Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

The caller-provided title, data set name, dump index name, symptom record, incident token, remote area, problem description area and storage list area all have the same requirements and restrictions as the control parameters.

Programming requirements
None.

Restrictions
The caller may not have any FRRs established.

An IEATDUMP cannot succeed when another process within the task is exclusively holding the SYSZTIOT enqueue. Instead, a SVC dump would probably occur.

Input register information
Before issuing the IEATDUMP macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the IEATDUMP macro, the caller does not have to place any information into any access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
Performance implications

None.

Syntax

The parameters DCB, DCBAD, and ASYNC=YES are no longer supported.

The IEATDUMP macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td></td>
<td>One or more blanks must precede IEATDUMP.</td>
</tr>
<tr>
<td>IEATDUMP</td>
<td>One or more blanks must follow IEATDUMP.</td>
</tr>
<tr>
<td>DSNAD=dsnad</td>
<td>dsnad: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>DSN=dsn</td>
<td>dsn: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>DDNAME=ddname</td>
<td>ddname: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,HDRAD=hdrad</td>
<td>hdrad: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,HDR=hdr</td>
<td>hdr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,IDX=idx</td>
<td>idx: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,SYMREC=symrec</td>
<td>symrec: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,SYMRECAD=symrecad</td>
<td>symrecad: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,REMOTE=remote</td>
<td>remote: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,REMOTEAD=remotead</td>
<td>remotead: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PROBDESC=sprobdesc</td>
<td>probdesc: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PROBDESCAD=sprobdescad</td>
<td>probdescad: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,LIST=list</td>
<td>list: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,LISTAD=listad</td>
<td>listad: RS-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
**IEATDUMP transaction dump**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SUBPLSTAD=subplstad</td>
<td>subplstad: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_SUBPLST=subplst</td>
<td>subplst: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_DSPLISTAD=dsplistad</td>
<td>dsplistad: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_DSPLIST=dsplist</td>
<td>dsplist: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_SDATA=DEFS</td>
<td>Default: SDATA=DEFS</td>
</tr>
<tr>
<td>_SDATA=ALLNUC</td>
<td></td>
</tr>
<tr>
<td>_SDATA=CSA</td>
<td></td>
</tr>
<tr>
<td>_SDATA=GRSQ</td>
<td></td>
</tr>
<tr>
<td>_SDATA=LPA</td>
<td></td>
</tr>
<tr>
<td>_SDATA=LSQA</td>
<td></td>
</tr>
<tr>
<td>_SDATA=NUC</td>
<td></td>
</tr>
<tr>
<td>_SDATA=RGN</td>
<td></td>
</tr>
<tr>
<td>_SDATA=SQA</td>
<td></td>
</tr>
<tr>
<td>_SDATA=SUM</td>
<td></td>
</tr>
<tr>
<td>_SDATA=SWA</td>
<td></td>
</tr>
<tr>
<td>_SDATA=TRT</td>
<td></td>
</tr>
<tr>
<td>_SDATA=PSA</td>
<td></td>
</tr>
<tr>
<td>_ASYNC=NO</td>
<td>Default: ASYNC=NO</td>
</tr>
<tr>
<td>_ECBAD=ecbad</td>
<td>ecbad: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_ECB=ecb</td>
<td>ecb: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>_PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>_PLISTVER=1</td>
<td></td>
</tr>
<tr>
<td>_MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>_MF=(L,list addr)</td>
<td>list addr: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td>_MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>_MF=(L,list addr,OD)</td>
<td></td>
</tr>
<tr>
<td>_MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>_MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
<tr>
<td>_MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
<tr>
<td>_MF=(M,list addr)</td>
<td></td>
</tr>
</tbody>
</table>
IEATDUMP transaction dump

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'.MF=(M,list addr,COMPLETE)'</td>
<td></td>
</tr>
<tr>
<td>'.MF=(M,list addr,NOCHECK)'</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters**

The parameters DCB, DCBAD, and ASYNC=YES are no longer supported, and are removed from this information.

The parameters are explained as follows:

**name**

An optional symbol, starting in column 1, that is the name on the IEATDUMP macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

**DSNAD=dsnad**

A required input parameter. The output dump data set should have the attributes of RECFM=FB and LRECL=4160.

**DSN=dsn**

A 2- to 101-character input area that contains the name pattern used to create the dump. The format of the area begins with a single byte specifying the length of the name pattern, which must not be greater than 100. The name pattern immediately follows that byte. The name pattern has a series of attributes: it is similar to that used by the operator DUMPDS NAME= parameter, except that &SEQ is not supported, and there is no default name pattern available; the use of system symbols is supported; and it must resolve to a valid data set name which can be allocated from the caller's task. When used with the REMOTE= parameter, the generated name must be unique for each requested address space (&JOBNAME is one recommended addition to the pattern to accomplish this).

In addition, IEATDUMP also recognizes the symbol &DS. (Dump Section) on the end of the name pattern. When present, IEATDUMP allocates the first data set for dumping, ending with “001”. If this runs out of disk space or uses up all 16 extents before the dump is completed, dumping will be continued to data sets with the same name, but ending in “002”, “003”, and so on, until the entire dump is written. Each of these data sets are allocated with a primary extent size of 500M and a secondary extent size of 500M, but it is possible to change these values by providing ACS routines that are driven by DFSMS.

Remember to combine all of the data sets into one data set by using IPCS COPYDUMP, before using IPCS to view the diagnostic data.
### IEATDUMP transaction dump

**To code:** Specify the RS-type address, or address in register (2) - (12), of a 2- to 101-character field.

**DDNAME=ddname**

An 8-character input field that is the name of the DD representing the data set that is to contain the dump. The DD must be allocated when IEATDUMP is invoked. The system will open this DD.

**To code:** Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

**,HDRAD=hdrad**

A required input parameter.

**,HDR=hdr**

A 4-byte field which contains the address of a parameter of the dump title. The format of the area is a single byte specifying the length of the title followed by the title itself.

**To code:** Specify the RS-type address, or address in register (2) - (12), of a pointer field.

**,HDR=hdr**

A 2- to 101-character input area that contains the dump title. The format of the area is a single byte specifying the length of the title followed by the title itself. The title has a maximum length of 100 characters.

**To code:** Specify the RS-type address, or address in register (2) - (12), of a 2- to 101-character field.

**,IDXAD=idxad**

An optional input parameter.

**,IDX=idx**

A 4-byte field which contains the address of a parameter of an area that contains the name of the dump index which is to contain information about the dump after the dump is written. The format of the area is a single byte specifying the length of the dump index data set name followed by the name itself. The data set must be an existing IPCS dump directory. The data set will be allocated from the caller’s address space.

**To code:** Specify the RS-type address, or address in register (2) - (12), of a pointer field.

**,IDX=idx**

A 2- to 45-character input area that contains the name of the dump index which is to contain information about the dump after the dump is written. The format of the area is a single byte specifying the length of the dump index data set name followed by the name itself. The name of the dump index data set has a maximum length of 44 characters. The data set must be an existing IPCS dump directory. The data set will be allocated from the caller’s address space.

**To code:** Specify the RS-type address, or address in register (2) - (12), of a 2- to 45-character field.

**,SYMRECAD=symrecad**

**,SYMREC=symrec**

An optional input parameter.
IEATDUMP transaction dump

,,SYMRECAD=symrecad
A 4-byte field which contains the address of a parameter of a valid
symptom record for DAE to use for dump suppression. This area is built
using SYMRBLD and mapped by ADSR. This area has a maximum length
of 1900 bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a
pointer field.

,,SYMREC=symrec
A parameter of a valid symptom record for DAE to use for dump
suppression. This area is built using SYMRBLD and mapped by ADSR.
This area has a maximum length of 1900 bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a
character field.

,,INTOKENAD=intokenad
,,INTOKEN=intoken
An optional input parameter.

,,INTOKENAD=intokenad
A 4-byte field which contains the address of a parameter of a 32-byte area
that contains an incident token previously built by the IEAINTKN macro.

To code: Specify the RS-type address, or address in register (2) - (12), of a
pointer field.

,,INTOKEN=intoken
A parameter of a 32-byte area that contains an incident token previously
built by the IEAINTKN macro.

To code: Specify the RS-type address, or address in register (2) - (12), of a
32-character field.

,,REMOTEAD=remotead
,,REMOTE=remote
An optional input parameter.

,,REMOTEAD=remotead
A 4-byte field which contains the address of an area that identifies other
address spaces to be dumped. This keyword is restricted to authorized
callers. The format of the area is described in the REMOTE parameter
which follows.

To code: Specify the RS-type address, or address in register (2) - (12), of a
pointer field.

,,REMOTE=remote
An optional character input area that can be a maximum of 1024 bytes
long, which identifies other address spaces to be dumped. This keyword is
restricted to authorized callers. The address spaces can be on the current
system and/or other systems in the sysplex. The area is mapped by the
IHASDRMT mapping macro. Through IHASDRMT, you can identify the
systems to be dumped and specify the content of the dumps on individual
systems. One can also specify that the following parameters on the
IEATDUMP macro be copied for the remote dumps requested: SDATA,
DSPLIST, and SUBPFLST. The area consists of:

• A 4-byte header, which indicates the total length of the area. The length
must include the four bytes of the header.
• Contents entry. Each entry consists of:
IEATDUMP transaction dump

ID  A 2-byte field, whose value identifies the content type. The values are declared by the constants with names beginning with SDRMT_IDCON in the IHASDRMT mapping.

Length  A 2-byte field that gives the length of the contents portion. The length must include the 2 bytes of this length field, plus the 2 bytes of the ID field.

Contents  A variable field that gives the contents identified in the ID field. The contents you can specify are the system names, job names, XCF group and member names, data space names, address space identifiers, SDATA options, storage ranges, subpools, and keys. Within the contents, the following items also support the use of wildcards:
- System name
- Job name
- XCF group name
- XCF member name
- Data space name and its qualifying job name
See wildcard support under the description of the SDUMPX macro.

To code: Specify the RS-type address, or address in register (2) - (12), of a character field.

,PROBDESCAD=probdescad
,PROBDESC=probdesc
An optional input parameter.

,PROBDESCAD=probdescad
A 4-byte field which contains the address of a parameter of an area that contains information describing the problem. This area has a maximum length of 1024 bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,PROBDESC=probdesc
A parameter of an area that contains information describing the problem. This area has a maximum length of 1024 bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a character field.

,LISTAD=listad
,LIST=list
An optional input parameter.

,LISTAD=listad
A 4-byte field which contains the address of a parameter of a list of starting and ending addresses of areas to be dumped. The high-order bit of the last ending address is set to 1; the high-order bit of all other addresses is 0. This area has a maximum length of 240 bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.
IEATDUMP transaction dump

\[ \text{LIST} = \text{list} \]
A parameter of a list of starting and ending addresses of areas to be dumped. The high-order bit of the last ending address is set to 1; the high-order bit of all other addresses is 0. This area has a maximum length of 240 bytes.

\textbf{To code}: Specify the RS-type address, or address in register (2) - (12), of a character field.

\[ \text{SUBPLSTAD} = \text{subplstad} \]
\[ \text{SUBPLST} = \text{subplst} \]
An optional input parameter.

\[ \text{SUBPLSTAD} = \text{subplstad} \]
A 4-byte field which contains the address of a parameter of a list of subpool numbers to be dumped. The first halfword is the number subpools in the list and must be on a fullword boundary. Each entry is two bytes.

\textbf{To code}: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

\[ \text{SUBPLST} = \text{subplst} \]
A parameter of a list of subpool numbers to be dumped. The first halfword is the number subpools in the list and must be on a fullword boundary. Each entry is two bytes.

\textbf{To code}: Specify the RS-type address, or address in register (2) - (12), of a character field.

\[ \text{DSPLISTAD} = \text{dsplistad} \]
\[ \text{DSPLIST} = \text{dsplist} \]
An optional input parameter.

\[ \text{DSPLISTAD} = \text{dsplistad} \]
A 4-byte field which contains the address of a parameter of a list of data space storage to be dumped. The first word is the total size of the DSPLIST. The next eight characters is the STOKEN of the data space to be dumped. A full word indicates the number of ranges to be dumped for that STOKEN. Then, 2 full words for each range, which are the starting and ending addresses of the range. More than one STOKEN may be specified per DSPLIST.

\textbf{To code}: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

\[ \text{DSPLIST} = \text{dsplist} \]
A parameter of a list of data space storage to be dumped. The first word is the total size of the DSPLIST. The next eight characters is the STOKEN of the data space to be dumped. A full word indicates the number of ranges to be dumped for that STOKEN. Then, 2 full words for each range, which are the starting and ending addresses of the range. More than one STOKEN may be specified per DSPLIST.

\textbf{To code}: Specify the RS-type address, or address in register (2) - (12), of a character field.

\[ \text{SDATA} = \text{DEFS} \]
\[ \text{SDATA} = \text{ALLNUC} \]
\[ \text{SDATA} = \text{CSA} \]
\[ \text{SDATA} = \text{GRSQ} \]
\[ \text{SDATA} = \text{LPA} \]
\[ \text{SDATA} = \text{LSQA} \]
IEATDUMP transaction dump

\textbf{SDATA=NUC}
- An optional parameter that specifies what system data should be provided in the transaction dump. No fetch-protected storage which is inaccessible in the caller’s key will be dumped. The default is SDATA=DEFS.

\textbf{SDATA=DEFS}
- The following SDATA options are included in the dump: LSQA, NUC, PSA, RGN, SQA, SUM, SWA, and TRT.

\textbf{SDATA=ALLNUC}
- All of DAT-on nucleus, including page-protected areas, and all of the DAT-off nucleus.

\textbf{SDATA=CSA}
- Common storage area and virtual storage for 64-bit addressable memory objects created using one of the following services:
  - IARV64 REQUEST=GETCOMMON,DUMP=LIKECSA
  - IARCP64 COMMON=YES,DUMP=LIKECSA
  - IARST64 COMMON=YES,TYPE=PAGEABLE

\textbf{SDATA=GRSQ}
- Global resource serialization (ENQ/DEQ/RESERVE) queues.

\textbf{SDATA=LPA}
- Link pack area for this job.

\textbf{SDATA=LSQA}
- Local system queue area and virtual storage for 64-bit addressable memory objects created using one of the following services:
  - IARV64 REQUEST=GETSTOR,DUMP=LIKELSQA
  - IARCP64 COMMON=NO,DUMP=LIKELSQA
  - IARST64 COMMON=NO

\textbf{SDATA=NUC}
- Non-page-protected areas of the DAT-on nucleus.

\textbf{SDATA=RGN}
- Entire private area and virtual storage for 64-bit addressable memory objects created using one of the following services:
  - IARV64 REQUEST=GETSTOR,DUMP=LKERGN
  - IARCP64 COMMON=NO,DUMP=LKERGN
  - IARST64 COMMON=NO

\textbf{SDATA=SQA}
- System queue area and virtual storage for 64-bit addressable memory objects created using one of the following services:
  - IARV64 REQUEST=GETCOMMON,DUMP=LIKESQA
  - IARCP64 COMMON=YES,DUMP=LIKESQA
  - IARST64 COMMON=YES,TYPE=FIXED
  - IARST64 COMMON=YES,TYPE=DREF

\textbf{SDATA=SUM}
- Requests the summary dump function.
IEATDUMP transaction dump

,SDATA=SWA
  Scheduler work area.

,SDATA=TRT
  System trace data.

,SDATA=PSA
  Prefixed save area.

One or more values may be specified for the SDATA parameter. If more than one value is specified, group the values within parentheses.

,ASYNC=NO
  An optional parameter that specifies whether the transaction dump should be taken synchronously. The default is ASYNC=NO.

,ASYNC=NO
  The transaction dump should be taken synchronously.

,ECBAD=ecbad
,ECB=ecb
  An optional input parameter.

,ECBAD=ecbad
  A 4-byte field which contains the address of a parameter of an ECB to be posted when the entire dump has been written. This area must be on a word boundary.

  To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,ECB=ecb
  A parameter of an ECB to be posted when the entire dump has been written. This area must be on a word boundary.

  To code: Specify the RS-type address, or address in register (2) - (12), of a 4-character field.

,RETCODE=retcode
  An optional output parameter into which the return code is to be copied from GPR 15.

  To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,RSNCODE=rsnocode
  An optional output parameter into which the reason code is to be copied from GPR 0.

  To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
  An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:
  • IMPLIED_VERSION, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
IEATDUMP transaction dump

- MAX, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

- 1, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

\[MF=S\]
\[MF=(L,list addr)\]
\[MF=(L,list addr,attr)\]
\[MF=(L,list addr,OD)\]
\[MF=(E,list addr)\]
\[MF=(E,list addr,COMPLETE)\]
\[MF=(E,list addr,NOCHECK)\]
\[MF=(M,list addr)\]
\[MF=(M,list addr,COMPLETE)\]
\[MF=(M,list addr,NOCHECK)\]

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms of IEATDUMP in the following order:

- Use IEATDUMP .MF=(M,list-addr,COMPLETE) specifying appropriate parameters, including all required ones.
- Use IEATDUMP .MF=(M,list-addr,NOCHECK), specifying the parameters that you want to change.
- Use IEATDUMP .MF=(E,list-addr,NOCHECK), to execute the macro.
**IEATDUMP transaction dump**

**list addr**
The name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this can be an RS-type address or an address in register (1)-(12).

**attr**
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

**COMPLETE**
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

**NOCHECK**
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

**ABEND codes**
None.

**Return and reason codes**
When the IEATDUMP macro returns control to your program:
- GPR 15 (and retcode, if you coded RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, if you coded RSNCODE) contains a reason code.

- X'00000000'
  A complete dump was written.

- X'00000004'
  A partial dump was written.

- X'00000008'
  No dump was written.

- X'0000000C'
  Internal processing error. No dump was written.

- X'00000010'
  Unexpected return code from IEAVAD00.

**Table 52. Return and Reason Codes for the IEATDUMP Macro**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00000000</td>
<td>Meaning: A complete dump was written.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>00000004</td>
<td>00000001</td>
<td>Meaning: The dump was truncated because the data set was too small.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Reissue IEATDUMP with a larger data set or use the DSN</td>
</tr>
<tr>
<td>00000004</td>
<td>00000002</td>
<td>Meaning: Contention detected when attempting to set tasks in the address space non-dispatchable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Data in dump may be inconsistent. Reissue IEATDUMP.</td>
</tr>
</tbody>
</table>
### IEATDUMP transaction dump

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00000004    | 00000003    | **Meaning:** Unable to add dump data set to dump index.  
**Action:** Verify that the dump index specified on the IDX parameter is correct and reissue IEATDUMP. |
| 00000004    | 00000004    | **Meaning:** Unable to allocate transaction dump data set.  
**Action:** See allocation failure messages. Reissue IEATDUMP. |
| 00000004    | 00000006    | **Meaning:** Maximum amount of dump sections reached (999).  
**Action:** Dump less memory, or use ACS routines to increase the size of the data sets. Reissue IEATDUMP. |
| 00000004    | 00000007    | **Meaning:** The system has filled one of the range tables.  
**Action:** Dump less memory. If the problem still exists, contact the IBM Support Center. |
| 00000004    | 00000008    | **Meaning:** The data space used for the IEATDUMP has been filled. No more then 2 gigabytes of data can be collected.  
**Action:** Do one of the following:  
- Remove unnecessary dump options.  
- Specify smaller memory ranges.  
- Use the extended data set support by either:  
  - Preallocating a large capacity data set and use the DDNAME parameter.  
  - Refer to the use of the &DS symbol for the DSN parameter’s data set name pattern. |
| 00000008    | 00000001    | **Meaning:** The address of the transaction dump parameter list was zero.  
**Action:** Ensure register 1 is non-zero when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 00000002    | **Meaning:** The dump was suppressed by CHNGDUMP.  
**Action:** Issue CHNGDUMP SET,SYSDUMP or CHNGDUMP RESET,SYSDUMP. Reissue IEATDUMP. |
| 00000008    | 00000003    | **Meaning:** The dump was suppressed by SLIP.  
**Action:** Delete SLIP trap with SLIP DEL command. Reissue IEATDUMP. |
| 00000008    | 00000004    | **Meaning:** The ALET for the transaction dump parameter list was not valid.  
**Action:** Ensure that access register 1 has a valid ALET when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 00000005    | **Meaning:** The transaction dump parameter list was not addressable.  
**Action:** Ensure that the entire transaction dump parameter list is addressable via register 1 (and access register 1 if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP. |
### Table 52. Return and Reason Codes for the IEATDUMP Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00000008    | 00000006    | **Meaning:** The transaction dump parameter list version number was not valid.  
**Action:** Ensure the transaction dump request was built using the IEATDUMP macro for the system on which the dump was requested. Reissue IEATDUMP. |
| 00000008    | 00000007    | **Meaning:** The length of the transaction dump parameter list did not match the parameter list version number.  
**Action:** Ensure the transaction dump request was built using the IEATDUMP macro for the system on which the dump was requested. Reissue IEATDUMP. |
| 00000008    | 00000008    | **Meaning:** No DDNAME, DSN(AD), or DSP_STOKEN was specified.  
**Action:** Reissue IEATDUMP with the DDNAME, DSN(AD) or DSP_STOKEN keyword. |
| 00000008    | 00000009    | **Meaning:** Both DDNAME and DSN(AD) keywords were specified.  
**Action:** Reissue IEATDUMP with either the DDNAME or DSN(AD) keyword. |
| 00000008    | 0000000C    | **Meaning:** The ALET for the DSN(AD) keyword was not valid.  
**Action:** Ensure that the access register for the DSN(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 0000000D    | **Meaning:** The DSN(AD) was not addressable.  
**Action:** Ensure that the entire DSN(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 0000000E    | **Meaning:** No HDR(AD) keyword was specified.  
**Action:** Reissue IEATDUMP with the HDR(AD) keyword. |
| 00000008    | 0000000F    | **Meaning:** The ALET for the HDR(AD) keyword was not valid.  
**Action:** Ensure that the access register for the HDR(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 00000010    | **Meaning:** The HDR(AD) was not addressable.  
**Action:** Ensure that the entire HDR(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 00000011    | **Meaning:** The specified HDR(AD) was longer than 100 characters.  
**Action:** Reissue IEATDUMP with a shorter header. |
| 00000008    | 00000012    | **Meaning:** The ALET for the IDX(AD) keyword was not valid.  
**Action:** Ensure that the access register for the IDX(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP. |
### Table 52. Return and Reason Codes for the IEATDUMP Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000008</td>
<td>00000013</td>
<td>The IDX(AD) was not addressable.</td>
<td>Ensure that the entire IDX(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000014</td>
<td>The IDX(AD) keyword did not specify a valid data set name after symbol substitution.</td>
<td>Reissue IEATDUMP with an IDX keyword that resolves to a valid dump index data set name.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000015</td>
<td>The ALET for the SYMREC(AD) keyword was not valid.</td>
<td>Ensure that the access register for the SYMREC(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000016</td>
<td>The SYMREC(AD) was not addressable.</td>
<td>Ensure that the entire SYMREC(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000017</td>
<td>The specified SYMREC(AD) was not valid. Either ADSRID not set to ‘SR’ or primary symptom string offset or length not initialized.</td>
<td>Reissue IEATDUMP with a valid symptom record.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000018</td>
<td>The ALET for the INTOKEN(AD) keyword was not valid.</td>
<td>Ensure that the access register for the INTOKEN(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000019</td>
<td>The INTOKEN(AD) was not addressable.</td>
<td>Ensure that the entire INTOKEN(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.</td>
</tr>
<tr>
<td>00000008</td>
<td>0000001A</td>
<td>The ALET for the REMOTE(AD) keyword was not valid.</td>
<td>Ensure that the access register for the REMOTE(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.</td>
</tr>
<tr>
<td>00000008</td>
<td>0000001B</td>
<td>The REMOTE(AD) was not addressable.</td>
<td>Ensure that the entire REMOTE(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.</td>
</tr>
<tr>
<td>00000008</td>
<td>0000001C</td>
<td>The specified REMOTE(AD) was not valid.</td>
<td>Reissue IEATDUMP with a valid remote area.</td>
</tr>
<tr>
<td>00000008</td>
<td>0000001D</td>
<td>The ALET for the LIST(AD) keyword was not valid.</td>
<td>Ensure that the access register for the LIST(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.</td>
</tr>
</tbody>
</table>
### IEATDUMP transaction dump

#### Table 52. Return and Reason Codes for the IEATDUMP Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000008</td>
<td>0000001E</td>
<td><strong>Meaning</strong>: The LIST(AD) was not addressable.</td>
<td><strong>Action</strong>: Ensure that the entire LIST(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.</td>
</tr>
<tr>
<td>00000008</td>
<td>0000001F</td>
<td><strong>Meaning</strong>: The specified LIST(AD) was not valid. A range in the storage list had a start address greater than its ending address.</td>
<td><strong>Action</strong>: Reissue IEATDUMP with a valid storage list.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000020</td>
<td><strong>Meaning</strong>: The dump was rejected because the caller’s authorization was insufficient for requested function(s).</td>
<td><strong>Action</strong>: Verify authorization and requested functions. Reissue IEATDUMP.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000021</td>
<td><strong>Meaning</strong>: The DSN(AD) keyword did not specify a valid data set name after symbol substitution.</td>
<td><strong>Action</strong>: Reissue IEATDUMP with a DSN keyword that resolves to a valid dump data set name.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000022</td>
<td><strong>Meaning</strong>: The DSN(AD) keyword specified a data set name that was too long.</td>
<td><strong>Action</strong>: Reissue IEATDUMP with a DSN(AD) keyword that resolves to a shorter dump data set name.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000023</td>
<td><strong>Meaning</strong>: The DSN(AD) keyword specified a data set name that contained a bad symbol.</td>
<td><strong>Action</strong>: Reissue IEATDUMP with a DSN(AD) keyword that does not contain bad symbols.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000024</td>
<td><strong>Meaning</strong>: Unable to create data space to capture transaction dump.</td>
<td><strong>Action</strong>: Remedy cause of DSPSERV CREATE failure or request transaction dump specifying DDNAME or including the &amp;DS. symbol in the DSN template.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000025</td>
<td><strong>Meaning</strong>: Unable to add transaction dump data space to access list.</td>
<td><strong>Action</strong>: Remedy cause of ALESERV ADD failure or request transaction dump specifying DDNAME. Reissue IEATDUMP.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000026</td>
<td><strong>Meaning</strong>: Unable to allocate transaction dump data set.</td>
<td><strong>Action</strong>: Look at allocation failure messages. Reissue IEATDUMP.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000027</td>
<td><strong>Meaning</strong>: The transaction dump was suppressed by DAE.</td>
<td><strong>Action</strong>: If you do not wish transaction dumps to be suppressed on an installation basis, issue the SET DAE=xx console command specifying an ADYSETxx member that does not specify SYSMDUMP(SUPPRESS). If you do not wish transaction dumps to be suppressed on an application basis, include the VRANODAE key in the VRADATA of your recovery routine. Reissue IEATDUMP.</td>
</tr>
</tbody>
</table>
### IEATDUMP transaction dump

Table 52. Return and Reason Codes for the IEATDUMP Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00000008    | 00000028    | **Meaning:** An error occurred writing the first record to the data space or dump data set.  
**Action:** Ensure the STOKEN and origin for the specified data space are correctly specified. Ensure that the specified DD is allocated when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 00000029    | **Meaning:** The ALET for the PROBDESC(AD) keyword was not valid.  
**Action:** Ensure that the access register for the PROBDESC(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 0000002A    | **Meaning:** The PROBDESC(AD) was not addressable.  
**Action:** Ensure that the entire PROBDESC(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 0000002B    | **Meaning:** The specified PROBDESC(AD) was not valid.  
**Action:** Reissue IEATDUMP with a valid problem description area. |
| 00000008    | 0000002C    | **Meaning:** The ALET for the SUBPLST(AD) keyword was not valid.  
**Action:** Ensure that the access register for the SUBPLST(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 0000002D    | **Meaning:** The SUBPLST(AD) was not addressable.  
**Action:** Ensure that the entire SUBPLST(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 0000002E    | **Meaning:** The specified SUBPLST(AD) was not valid. An invalid subpool was specified.  
**Action:** Reissue IEATDUMP with a valid subpool list. |
| 00000008    | 0000002F    | **Meaning:** The ALET for the DSPLIST(AD) keyword was not valid.  
**Action:** Ensure that the access register for the DSPLIST(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 00000030    | **Meaning:** The DSPLIST(AD) was not addressable.  
**Action:** Ensure that the entire DSPLIST(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP. |
| 00000008    | 00000031    | **Meaning:** The specified DSPLIST(AD) was not valid. An invalid data space was specified.  
**Action:** Reissue IEATDUMP with a valid data space list. |
| 00000008    | 00000032    | **Meaning:** The ALET for the ECB(AD) keyword was not valid.  
**Action:** Ensure that the access register for the ECB(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP. |
### Table 52. Return and Reason Codes for the IEATDUMP Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00000008    | 00000033    | **Meaning:** The ECB(AD) was not addressable.  
**Action:** Ensure that the entire ECB(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested.  
Reissue IEATDUMP. |
| 00000008    | 00000034    | **Meaning:** The specified ECB(AD) was not valid. The ECB was not on a fullword boundary.  
**Action:** Reissue IEATDUMP with an ECB. |
| 00000008    | 00000035    | **Meaning:** OPEN failed for the dump data set.  
**Action:** Determine why OPEN failed and reissue IEATDUMP. |
| 00000008    | 00000036    | **Meaning:** Dump data set has invalid block size.  
**Action:** Correct the block size and reissue IEATDUMP. |
| 00000008    | 00000037    | **Meaning:** The DSP_RECORDS@ field was not accessible.  
**Action:** Correct the problem and reissue IEATDUMP. |
| 00000008    | 00000038    | **Meaning:** The DCB parameter is not supported on IEATDUMP.  
**Action:** Remove the DCB parameter and reissue IEATDUMP. |
| 00000008    | 00000039    | **Meaning:** The ASYNC=YES is not supported on IEATDUMP.  
**Action:** Change to ASYNC=NO and reissue IEATDUMP. |
| 0000000B    | 0000003A    | **Meaning:** The &DS. symbol was found in the midst of the dump DSN name pattern.  
**Action:** Place the &DS symbol at the end of the DSN name pattern and reissue IEATDUMP. |
| 0000000B    | 0000003B    | **Meaning:** This IEATDUMP was not taken because another dump was already running in the address space.  
**Action:** None. |
| 0000000C    | 00000001    | **Meaning:** Unable to obtain storage for transaction dump from subpool 230 below the line.  
**Action:** Determine why storage is not available and reissue IEATDUMP. |
| 0000000C    | 00000002    | **Meaning:** Unable to establish recovery environment for transaction dump.  
**Action:** Determine why ESTAEX failed and reissue IEATDUMP. |
| 0000000C    | 00000003    | **Meaning:** Unable to obtain storage for transaction dump from subpool 239 above the line.  
**Action:** Determine why storage is not available and reissue IEATDUMP. |
| 0000000C    | 00000004    | **Meaning:** Unable to obtain storage for transaction dump from subpool 231 above the line.  
**Action:** Determine why storage is not available and reissue IEATDUMP. |
### IEATDUMP transaction dump

#### Table 52. Return and Reason Codes for the IEATDUMP Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0000000C    | 00000005    | **Meaning**: Unable to obtain storage for transaction dump from subpool 239 above the line.  
**Action**: Determine why storage is not available and reissue IEATDUMP. |
| 0000000C    | 00000006    | **Meaning**: Unable to obtain storage for transaction dump from subpool 239 above the line.  
**Action**: Determine why storage is not available and reissue IEATDUMP. |
| 0000000C    | 00000007    | **Meaning**: Unable to obtain storage for transaction dump from subpool 239 above the line.  
**Action**: Determine why storage is not available and reissue IEATDUMP. |
| 0000000C    | 00000008    | **Meaning**: Unable to obtain storage for transaction dump from subpool 250 above the line.  
**Action**: Determine why storage is not available and reissue IEATDUMP. |
| 0000000C    | 00000009    | **Meaning**: Unable to obtain storage for transaction dump from subpool 230 above the line.  
**Action**: Determine why storage is not available and reissue IEATDUMP. |
| 0000000C    | 0000000A    | **Meaning**: Unable to obtain storage for transaction dump from subpool 230 below the line.  
**Action**: Determine why storage is not available and reissue IEATDUMP. |
| 0000000C    | 0000000B    | **Meaning**: Unable to obtain storage for transaction dump from subpool 253 above the line.  
**Action**: Determine why storage is not available and reissue IEATDUMP. |
| 0000000C    | 000000FF    | **Meaning**: IEATDUMP’s recovery received control. One possible reason is that the SYSZTIOT enqueue is being held exclusively by another process running under this task. It is not possible for the IEATDUMP to successfully complete.  
**Action**: The assistance of a system programmer is needed for associated SVC dumps. In the case of a SYSZTIOT enqueue, the problem is not in IEATDUMP processing. The diagnosis of any issues requires data collection using SLIP and/or SDUMPX, and not IEATDUMP. |
| 00000010    | xxxxxxxx    | **Meaning**: Unexpected return code from IEAVAD00. Return code from IEAVAD00 returned as reason code.  
**Action**: Inform the system programmer. |

#### Examples

An example using DSN:

```assembler
IEATDUMP DSN=DUMPDSN,HDR=DUMPTTL2
.
.
.
DUMPDSN DC AL1(EZ-S2)
S2  DC 'HLQ.TDUMP.D&&YYMDD..T&&HHMMSS..&&SYSNAME..&&JOBNAME.'
```

582  z/OS V2R1.0 MVS Authorized Assembler Services Reference EDT-IXG
IEATDUMP transaction dump

E2 EQU *
DUMPTTL2 DC AL1(E3-S3)
S3 DC C'IEADUMP TO AUTOMATICALLY ALLOCATED DATA SET'
E3 EQU *
IEATDUMP transaction dump
Chapter 46. IEATEDS - Timed event data services

Description

IEATEDS provides timed event data services.

IEATEDS allows the user to record events to a Timed Event Data Table to provide information that will help determine flow and performance. Each event is time stamped and includes data provided by the caller and additional data collected by the service. A REXX exec is also provided to obtain a formatted report of the events.

To use the timed event data service:
1. Invoke IEATEDS with REQUEST=REGISTER to obtain and initialize a Timed Event Data Table. The size of the Timed Event Data Table is determined by the MaxEvents argument. Note that the Timed Event Data Table will not wrap. The REGISTER service will provide a Timed Event Data Token as output which will identify the newly created Timed Event Data Table on subsequent IEATEDS requests.

2. Invoke IEATEDS with REQUEST=RECORD, passing the Timed Event Data Token and other arguments, including up to 16 bytes of user data. Several RECORD requests may be made throughout the code to understand the flow and performance. Once the maximum number of events has been recorded, subsequent requests will be ignored.

3. Execute REXX exec IEAVFTED to output the Timed Event Data Report to a data set. The Timed Event Data Report parameters are described below and the format of the output is described in the IEATEDS macro example section. Note that IEAVFTED does not clear the Timed Event Data Table. Thus, IEAVFTED may be executed at any time to produce an up-to-date report containing all of the events that have been recorded so far.

Timed Event Data Report

The IBM supplied IEAVFTED REXX exec is used to produce a Timed Event Data Report in either a TSO or IPCS environment. When run under TSO, the user must specify either a pre-allocated data set or a z/OS UNIX file in which to place the report. The dataset option requires the user to allocate a data set with an LRECL of 512 and a RECFM of V or VB. The z/OS UNIX file option requires the TSO environment to have an OMVS segment. When run under IPCS, the Timed Event Data Report will normally be displayed within IPCS. An example is provided in the IEATEDS example section that shows how to have IPCS place the Timed Event Data Report into a pre-allocated data set.

The Timed Event Data Report will consist of two sections, the first section consisting of human readable text, and the second section consisting of spreadsheet data (unless the NOSS parameter, described below, is specified). Note that the IPL Statistics Table (IPST) will also be placed into the Timed Event Data Report.

The IEAVFTED REXX exec is a compiled REXX program which requires the full REXX compiler run-time libraries (at least REXX LIBR BASE MVS FMID HWJ9140) installed before attempting to use IEAVFTED. Note that IEAVFTED will not work with the REXX Alternate Runtime Library z/OS Base HWJ9143.
The IEAVFTED code resides in data set SYS1.SBLSCL10. IEAVFTED must be run from a data set with an LRECL of 80 and a RECFM of F or FB.

The following describes the required and optional parameters for IEAVFTED:

- **DATASET(output_data_set)** or **DA(output_data_set)** is used to specify the name of the pre-allocated data set where the Timed Event Data Report will be written when IEAVFTED is run under TSO. The name must be fully qualified and the data set must have an LRECL of 512 with a RECFM of V or VB. Note that one and only one of DATASET(output_data_set), DA(output_data_set), or PATH(z/OS UNIX file) must be specified when IEAVFTED is run under TSO. Neither DATASET(output_data_set) nor DA(output_data_set) is allowed when IEAVFTED is run under IPCS (an example is provided in the IEATEDS macro example section that shows how to have IPCS place the Timed Event Data Report into a pre-allocated data set).

- **PATH(z/OS UNIX file)** is used to specify the name of a z/OS UNIX file where the Timed Event Data Report will be written when IEAVFTED is run under TSO. A z/OS UNIX file is created along with its directories with the authority options of 770. Note that one and only one of DATASET(z/OS UNIX file), DATASET(output_data_set), or DA(output_data_set) must be specified when IEAVFTED is run under TSO. PATH(z/OS UNIX file) is not allowed when IEAVFTED is run under IPCS as IPCS does not directly support putting output to a z/OS UNIX file.

- **IPCSDA(input_data_set)** is an optional specification that is used only with a special IEAVFTED invocation that will convert the spreadsheet data into a proper format in order to import it into a spreadsheet program. This is needed for the case where the Timed Event Data Report was obtained under IPCS using the example technique described in the IEATEDS macro example section. In that example technique, the output data set needs to be pre-allocated with an LRECL of 255 to satisfy IPCS, but in order to import the spreadsheet data into a spreadsheet program, the spreadsheet data needs to be in a 512 character data set or in a z/OS UNIX file. Invoking IEAVFTED with IPCSDA(input_data_set) does not produce a new report, but instead extracts the spreadsheet data contained in IPCSDA(input_data_set) and places it into the required data set as specified by one and only one of DATASET(output_data_set), DA(output_data_set), or PATH(z/OS UNIX file). Parameters SS, NOSS, Component(component_name), and Comp(component_name) are ignored when IPCSDA is specified. Note that 'input_data_set' and 'output_data_set' must be different data set names.

- **Component(component_name)** or Comp(component_name) is optional and is used as a filter to obtain Timed Event Data Table information for a specific component. Each Timed Event Data Table is created with the component name specified as CompName on the IEATEDS REGISTER request. When component(component_name) is specified, the Timed Event Data Report will include Timed Event Data Table information only for those Timed Event Data Tables with a matching component name. Note that if the component_name value contains blanks, it must be enclosed within quotes. The component_name value is not case sensitive and will be converted to uppercase. Also, Component(IPST) can be used to obtain only the IPL Statistics Table. When neither Component() nor Comp() is specified, all Timed Event Data Tables for all components are included in the report. Note that Component() and Comp() are ignored when IPCSDA is specified.

- **SS(char)** is optional and is used to specify the character delimiter for the spreadsheet fields. The default character is a semicolon (;). Note that SS(char) is ignored when NOSS or IPCSDA are specified.
• NOSS is optional and is used to cause the spreadsheet data to be omitted from 
the Timed Event Data Report. Note that NOSS is ignored when IPCSDA is 
specified.
• HELP or ? is optional and is used to get a description of IEAVFTED and its 
parameters.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state. Any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any primary, any home, and any secondary address space</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>The caller may hold a local lock, or a local lock and the CMS lock.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

The caller must include the IHAPSA, CVT, IHAECVT, and IHATEDS macros. Note 
that the IHATEDS macro has equate symbols for the return and reason codes, and 
for the length of the WorkArea.

Restrictions

None

Input register information

Before issuing the IEATEDS macro, the caller must ensure that general register 13 
contains the address of a 216 byte save area. The save area must be in primary 
storage in the first 2G of storage. The caller does not have to place any information 
into any other general purpose register (GPR) unless using it in register notation 
for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR15 is not zero</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
Unchanged

Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

```
IEATEDS REQUEST=REGISTER parameters-1

WORKAREA=workarea

RETCODE=retcode

RSNCODE=rsncode

MF=S

MF=(L,list addr,attr,COMPLETE)

parameters-1

COMPNAME=compname

MAXEVENTS=maxevents

TEDTOKEN=tedtoken

parameters-2

TEDTOKEN=tedtoken

EVENTTYPE=START

EVENTTYPE=MID

EVENTTYPE=END

EVENTTHREAD=eventthread

EVENTDESC=eventdesc

MODNAME=modname

MODLEVEL=modlevel
```
Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEATEDS macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,COMPNAM=compname
When REQUEST=REGISTER is specified, a required input parameter that specifies the component name that is registering. A mixed case value is supported.

To code: Specify the RS-type address, or address in register (2) - (12), of a 32-character field.

,EVENTDESC=eventdesc
When REQUEST=RECORD is specified, a required input parameter that is used to describe the event. A mixed case value is supported.

To code: Specify the RS-type address, or address in register (2) - (12), of a 32-character field.

,EVENTTHREAD=eventthread
When REQUEST=RECORD is specified, a required input parameter that is used to provide an association for a series of events consisting of a start event, zero or more mid events, and an end event. This can be any character or hex value that the component finds useful. Using a unique value for each series of associated start, mid, and end events will help in understanding the flow and timing of the events. The Timed Event Data Report will include both the hex and the EBCDIC values for the thread. See the macro example section for more information regarding the Timed Event Data Report.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,EVENTTYPE=START
,EVENTTYPE=MID
,EVENTTYPE=END
When REQUEST=RECORD is specified, a required parameter that indicates the type of event to record.

,EVENTTYPE=START
The event is the start of a series of events.

,EVENTTYPE=MID
The event is one of a series of events. This mid event is matched to the start event whose EventThread matches the EventThread provided on this request.

,EVENTTYPE=END
The event is the last of a series of events. This end event is matched to the start event whose EventThread matches the EventThread provided on this request.

,MAXEVENTS=maxevents
When REQUEST=REGISTER is specified, a required input parameter that
specifies the maximum number of events that will be recorded. This value will be used to determine the amount of storage to be allocated for the Timed Event Data Table for recording the events. The Timed Event Data Table size will be capped at 2M bytes, meaning that the value specified for MaxEvents will be reduced as necessary to a value where the Timed Event Data Table will be created within 2M bytes of storage. A return code and reason code will be returned if the MaxEvents was reduced, unless some other more serious error is returned.

Note also that all Timed Event Data Table storage is capped at 2G bytes, and any attempt to REGISTER once the 2G limit is reached will be rejected with a return code and reason code. No attempt will be made to reduce the MaxEvents in order to build a Timed Event Data Table any smaller than 2M just to get it to fit into the remaining storage that is near the 2G limit. Note also that there may be additional system controls or environmental conditions that limit this size to something smaller than 2G bytes.

The number of events that will fit into a 2M Timed Event Data Table is a function of the size of each entry. Since the entry size could grow over time, the number of events that will fit could be reduced in the future. This makes it difficult to accurately state the maximum number of events a Timed Event Data Table will hold, but a maximum of at least 2000 events is guaranteed.

Note that any attempt to record events beyond the resultant MaxEvents will not be recorded, but will be counted as an overflow count to assist in determining whether the number of RECORD requests should be decreased or MaxEvents should be increased (if not already at or above the maximum for a 2M Timed Event Data Table). The overflow count, requested MaxEvents, resultant MaxEvents, and the size of the resultant Timed Event Data Table can be found in the Timed Event Data Report which is described in the macro example section.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field, or specify a literal decimal value.

\[\text{MF=}\text{S}\]
\[\text{MF=}\text{L}\text{, list addr}\]
\[\text{MF=}\text{L}\text{, list addr, attr}\]
\[\text{MF=}\text{L}\text{, list addr, 0D}\]
\[\text{MF=}\text{E}\text{, list addr}\]
\[\text{MF=}\text{E}\text{, list addr, COMPLETE}\]

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,MODLEVEL=modlevel
When REQUEST=RECORD is specified, a required input parameter that specifies the module level that is recording this event.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,MODNAME=modname
When REQUEST=RECORD is specified, a required input parameter that specifies the module name that is recording this event.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,PLISTVER=IMPLIED_VERSION, PLISTVER=MAX, PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- IMPLIED_VERSION, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- MAX, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- 0, if you use the currently available parameters.

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0

REQUEST=REGISTER
REQUEST=RECORD
A required parameter that indicates which service to perform.
**REQUEST=REGISTER**

Register the user for the timed event data service.

**REQUEST=RECORD**

Record the timed event data.

**RETCODE=retcode**

An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) or (15), (GPR15), (REG15), or (R15).

**RSN=rsncode**

An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2) - (12), (00), (GPR0), (GPR00), (REG0), (REG00), or (R0).

**TEDTOKEN=tedtoken**

When REQUEST=REGISTER is specified, a required output parameter, whose returned value must be provided as input on subsequent REQUEST=RECORD calls.

To code: Specify the RS-type address, or address in register (2) - (12), of a 16-character field.

**TEDTOKEN=tedtoken**

When REQUEST=RECORD is specified, a required input parameter that is used to identify the timed event data collection to which this event shall be placed. This must be the TedToken that was returned as output from the Register request.

To code: Specify the RS-type address, or address in register (2) - (12), of a 16-character field.

**USERDATA=userdata**

**USERDATA=NO_USERDATA**

When REQUEST=RECORD is specified, an optional input parameter consisting of a comma delimited list of one or more variable names with a combined length (determined using L’varname for each variable) that does not exceed a total of 16 bytes. UserData may be any data that the user finds helpful in understanding the timed events. Note that the values must be simple items - for example, using substringed references is not allowed. The default is NO_USERDATA.

One or more values may be specified for the USERDATA parameter. If more than one value is specified, group the values within parentheses.

To code: Specify the RS-type address of a character field.

**WORKAREA=workarea**

A required input parameter that specifies a work area on a double word boundary to be used by the timed event data service. The work area must be of size IEATEDS_WorkAreaSize (in macro IHATEDS) and must reside within the first 2 GB of primary storage in any key.

To code: Specify the RS-type address of a character field.
**ABEND codes**

0C2  **Meaning:** Privileged-operation exception. A problem state caller attempted to use IEATEDS.

**Action:** Get into supervisor state before invoking the function, perhaps by the MODESET macro.

**Return and reason codes**

When the IEATEDS macro returns control to your program:

- GPR 15 (and retcode, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsnocode, when you code RSNCODE) contains a reason code.

Macro IHATEDS provides equate symbols for the return and reason codes. Note that the return and reason codes described below are hexadecimal values.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code.

<table>
<thead>
<tr>
<th>Table 53. Return and reason codes for the IEATEDS macro</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Return code</strong></td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Chapter 46. IEATEDS - Timed event data services  593
<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason code</th>
<th>Equate symbol meaning and action</th>
</tr>
</thead>
</table>
| 4           | 00000401    | Equate symbol: IEATEDSRsn_TedTableFull  
**Meaning:** An IEATEDS RECORD request was unable to place the new entry into the caller’s Timed Event Data Table because the Timed Event Data Table is full. The Timed Event Data Table overflow count has been incremented.  
**Action:** Try increasing the value specified for MaxEvents on the IEATEDS REGISTER request. Note, however, that there is a limit of 2M bytes of storage for each Timed Event Data Table, and an overall limit of 2G bytes of storage for all of the Timed Event Data Tables in the system. You can execute the Timed Event Data Report REXX exec to examine the overflow count to determine the number of additional entries required to allow all of the RECORD requests to succeed. The Timed Event Data Report will also show the requested MaxEvents, the resultant MaxEvents, and the size of the Timed Event Data Table. If the size of the Timed Event Data Table is already at the 2M byte limit, try reducing the number of IEATEDS RECORD requests. |
| 4           | 00000402    | Equate symbol: IEATEDSRsn_MaxEventsReduced  
**Meaning:** An IEATEDS REGISTER request reduced the specified MaxEvents to allow the Timed Event Data Table to be built within the 2M-byte storage limit.  
**Action:** Execute the Timed Event Data Report REXX exec and examine the overflow count to determine whether the reduced MaxEvents has resulted in some RECORD requests not being recorded. If so, consider reducing the number of RECORD requests to a value that is no greater than the reduced MaxEvents (shown in the report as the resultant MaxEvents). |
| 8           | —           | Equate symbol: IEATEDSRc_InvParm  
**Meaning:** IEATEDS request specified parameters that are not valid.  
**Action:** Refer to the action provided with the specific reason code. |
| 8           | 00000801    | Equate symbol: IEATEDSRsn_BadTedToken  
**Meaning:** An IEATEDS RECORD request supplied a TedToken value that was unable to locate a valid Timed Event Data Table. The RECORD request was not completed.  
**Action:** Ensure that the TedToken returned from the REGISTER request is not corrupted and is provided on the subsequent IEATEDS RECORD requests. Note that this error will also occur in the case of a Timed Event Data Table being corrupted or the storage becoming inaccessible. Execute the Timed Event Data Report REXX exec (described in the macro example section) to see whether the Timed Event Data Table in question was able to be located. If so, then the problem is with the specified TedToken. |
| C           | —           | Equate symbol: IEATEDSRc_Env  
**Meaning:** Environmental error  
**Action:** Refer to the action provided with the specific reason code. |
### Table 53. Return and reason codes for the IEATEDS macro (continued)

<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason code</th>
<th>Equate symbol meaning and action</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>00000C01</td>
<td>Equate symbol: IEATEDSRSn_NoTedTableStorage</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> An IEATEDS REGISTER request was unable to obtain storage for the Timed Event Data Table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Ensure that the system has enough above the bar common storage to satisfy a request for a Timed Event Data Table of the size being requested. Note that the maximum size allowed for a Timed Event Data Table is 2M bytes, and the maximum size for all Timed Event Data Table storage to 2G bytes. Note also that there may be additional system controls or environmental conditions that limit this size to something smaller than 2G bytes.</td>
</tr>
<tr>
<td>C</td>
<td>00000C02</td>
<td>Equate symbol: IEATEDSRSn_NoTedVectorTableStorage</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> An IEATEDS REGISTER request was unable to obtain storage for a Timed Event Data Vector Table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Ensure that the system has enough above the bar common storage to satisfy a request for a Timed Event Data Vector Table which has a size of 4k.</td>
</tr>
<tr>
<td>10</td>
<td>—</td>
<td>Equate symbol: IEATEDSRSn_CompError</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> Unexpected failure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Contact your system programmer.</td>
</tr>
<tr>
<td>10</td>
<td>00010001</td>
<td>Equate symbol: IEATEDSRSn_UnexpectedError</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> An IEATEDS REGISTER or RECORD request had an unexpected error. The REGISTER of RECORD request completion status is unknown.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> The system programmer should gather any diagnostic information that was produced and contact IBM support.</td>
</tr>
</tbody>
</table>

### Examples

The following is an example of invoking IEATEDS to:

- Invoke IEATEDS to REGISTER
- Invoke IEATEDS to RECORD a Start event
- Invoke IEATEDS to RECORD a Mid event
- Invoke IEATEDS to RECORD an End event

The code is as follows.

```
TITLE 'Sample code to register/record Timed Event Data'
TEDSAMPL CSECT
TEDSAMPL AMODE 31
TEDSAMPL RMODE ANY
/* START OF SPECIFICATIONS ******************************************
* *01* MODULE-NAME = TEDSAMPL
* *02* DESCRIPTIVE-NAME = Sample program to register and
* *01* DISCLAIMER =
* This sample source is provided for tutorial purposes
* only. A complete handling of error conditions has not
* been shown or attempted, and this source has not been
```
submitted to formal IBM testing. This source is
distributed on an 'as is' basis without any warranties
either expressed or implied.

**** END OF SPECIFICATIONS *****************************/

EJECT
BAKR R14,0  Save on stack, return using r14
BASR 12,0
USING START,R12
START EQU *
MODID ,

*****************************/

** START OF CODE
*************************************************************/

START
MODID ,

*************************************************************/

USING DynArea,R13

*************************************************************/

** Register for Timed Event Data Recording
*************************************************************/

IEATEDS Request=REGISTER, *
CompName="CL32'TheProduct', *
MaxEvents="F'64', *
TedToken=TedToken, *
WorkArea=TedWorkArea, *
RetCode=RetCode, *
RsnCode=RsnCode, *
MF=(E,MyTedPLD,COMPLETE)

* Place code to check return/reason codes here.

*************************************************************/

** Record Timed Event Data for Event Thread SAMPLE
*************************************************************/

LA R2,1
ST R2,DATA1
MVC DATA2(4),=CL4' RCD'
IEATEDS Request=RECORD, *
EventType=START, *
EventThread="CL8'SAMPLE', *
EventDesc="CL32'Timed Event Data sample', *
UserData=(Data1,Data2), *
ModName="CL8'TEDSAMPL', *
ModLevel="CL8'Level101', *
TedToken=TedToken, *
WorkArea=TedWorkArea, *
RetCode=RetCode, *
RsnCode=RsnCode, *
MF=(E,MyTedPLD,COMPLETE)

* Place code to check return/reason codes here.

*************************************************************/

** Record Mid Timed Event Data
*************************************************************/

LA R2,2
ST R2,DATA1
MVC DATA2(4),=CL4'XYZ1'
MVC DATA3(6),=CL6'FUNC 1'
IEATEDS Request=RECORD, *
EventType=MID, *
EventThread="CL8'SAMPLE', *
EventDesc="CL32'Before doing XYZ', *
UserData=(Data1,Data2,Data3), *
ModName="CL8'TEDSAMPL', *
ModLevel="CL8'Level101', *
TedToken=TedToken, *
WorkArea=TedWorkArea, *
RetCode=RetCode, *
RsnCode=RsnCode, *
MF=(E,MyTedPLD,COMPLETE)
TedToken=TedToken, *
WorkArea=TedWorkArea, *
RetCode=RetCode, *
RsnCode=RsnCode, *
MF=(E,MyTedPLD,COMPLETE)

* Place code to check return/reason codes here.
*
******************************************************************************
* Record Last Timed Event Data for this Thread
******************************************************************************
LA R2,3
ST R2,DATA1
MVC DATA2(4),=CL4'XYZ1'
MVC DATA3(6),=CL6'FUNC 2'
IEATEDS Request=RECORD, *
  EventType=END, *
  EventThread==CL8'SAMPLE', *
  EventDesc==CL32'After doing XYZ', *
  UserData=(Data1,Data2,Data3), *
  ModName==CL8'TEDSAMPL', *
  ModLevel==CL8'Level101', *
  TedToken=TedToken, *
  WorkArea=TedWorkArea, *
  RetCode=RetCode, *
  RsnCode=RsnCode, *
  MF=(E,MyTedPLD,COMPLETE)

* Place code to check return/reason codes here.
*
******************************************************************************
* Free Dynamic Area and Return
******************************************************************************
LA R0,DynAreaLen Length of Dynamic Area
STORAGE RELEASE,LENGTH=(R0),ADDR=(R13),SP=240
PR
******************************************************************************
* Dynamic Area
******************************************************************************
DynArea DSECT
SaveArea DS XL216
TedToken DS XL16 Timed Event Data Token
Data1 DS F One word
Data2 DS CL4 Four bytes
Data3 DS CL6 Six Bytes
Data4 DS CL2 Two Bytes
RetCode DS F Return Code from Timed Event x
Data Service
RsnCode DS F Reason Code from Timed Event x
Data Service
DS OD Align TedWorkArea on dbl word
TedWorkArea DS XL(IEATEDS_WORKAREASIZE)
IEATEDS MF=(L,MyTedPLD)
DynAreaLen EQU *-DynArea Length of DynArea
IHATEDS Constants and ret/rsn codes
******************************************************************************
* REGISTER EQUATES
******************************************************************************
SPACE 1
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
Timed Event Data Report example invocations

Example invocations for TSO: The following example invocation will produce a Timed Event Data Report with all components with spreadsheet data with the default spreadsheet delimiter of a semicolon. IBM may request that this data be sent to IBM for analysis. Note that output_data_set must be the name of a pre-allocated data set with an LRECL of 512 and a RECFM of V or VB.

IEAVFTED DA('output_data_set')

The following example invocation will produce a Timed Event Data Report for component ABC with spreadsheet data and with a spreadsheet delimiter of a question mark. Note that the component name is not case sensitive.

IEAVFTED DA('output_data_set(member)') COMPONENT(ABC) SS(?)

The following invocation will produce a Timed Event Data Report with all components without the spreadsheet data:

IEAVFTED DA('output_data_set') NOSS

The following invocation will produce a Timed Event Data Report to a z/OS UNIX file for component ABC with spreadsheet data: Note that the file name is case sensitive, the directories need not exist and COMP is abbreviated for COMPONENT.

IEAVFTED PATH('/usr/ted_data/Performance_Data_For_System_XYZ')

The following example JCL will run the IEAVFTED REXX exec in the TSO background. Note that the SYSEXEC data set must have an LRECL of 80 and a RECFM of F or FB.

//IEAVFTED JOB '123456,?'
// 'name',REGION=0M,
// MSGLEVEL=(1,1),CLASS=J,NOTIFY=name,
// MSGCLASS=H
//IEAVFTED EXEC PGM=IKJEFT01,ROLL=(NO,NO),DYNAMNBR=400,REGION=0M
//*****************************************************************************
// Notes:
//** - The REXX compiler run-time libraries must be installed.
//**
//** - The data set containing the IEAVFTED exec
//** (SYS1.SBLSC10) must have an LRECL of 80 and a RECFM of F or FB.
//*****************************************************************************
//SYSEXEC DD DISP=SHR,DSN=SYS1.SBLSC10
//SYSPRINT DD SYSOUT=*
The following example JCL is a started procedure that will run the IEAVF TED REXX exec in the TSO background. Required and optional parameters for IEAVF TED are passed via the PARM keyword. See "Timed Event Data Report" on page 585 for the descriptions for the required and optional IEAVF TED parameters.

An example of a started procedure invocation using system symbolics as parameters:
S TEDRPT,PARM='PATH(/usr/ted/ted_report) COMP(ABC)'

An example of a started procedure invocation specifying a TSO dataset and having NOSS (no spreadsheet data).
S TEDRPT,PARM='da(console.mttr.output(sample)) noss'

Example IPCS invocations: The following example will produce a Timed Event Data Report with all components with spreadsheet data with a spreadsheet delimiter of a semicolon (the default). The output will be put into a data set that must be pre-allocated with an LRECL of 255 and a RECFM of V or VB. Note that IPCS only supports a maximum of 255 characters for the data set, but if the spreadsheet data needs to be imported into a spreadsheet program, then an additional procedure (described below) must be performed to extract the spreadsheet data from the 255 character data set and place it into either a pre-allocated 512 character data set or a z/OS UNIX file. Note that no data will be truncated or lost with the 255 character data set. The ALTLIB statement tells IPCS where to find the IEAVF TED exec. Note that the data set containing IEAVF TED (SYS1.SBLSCLI0) must have an LRECL of 80 and a RECFM of F or FB.

Create an output dsn (LRECL of 255, RECFM of V or VB)
TSO ALLOC F(IPCSPRINT) DA(dsn) SHR REUS
IP ALTLIB ACTIVATE APPL(EXEC) DA('SYS1.SBLSCLI0')
IP SETDEF PRINT NOTERM
IP IEAVF TED
IP CLOSE PRINT
IP SETDEF TERM NOPRINT
Data set dsn now contains the Timed Event Data Report

If the spreadsheet data needs to be imported into a spreadsheet program then one of the following IEAVF TED invocations must be issued from TSO. In the first example, the output_data_set must be pre-allocated with an LRECL of 512 and a RECFM of V or VB. In the second example, the output will go to a z/OS UNIX file. In both examples, the input_data_set is the 255 character data set from the above procedure for IPCS. IEAVF TED will extract the spreadsheet data from input_data_set, convert it into the proper format, and write it to the 512 character
output_data_set or to the z/OS UNIX file, either of which can then be downloaded or FTP’d and imported into a spreadsheet program.

IEAVFTED IPCSDA('input_data_set') DA('output_data_set')

...or...

IEAVFTED IPCSDA('input_data_set') PATH('z/OS UNIX file')

**Formatted Timed Event Data Report:** The following example Timed Event Data Report was obtained by running the example TEDSAMPL assembler program described above, and then invoking IEAVFTED as follows:

IEAVFTED PATH(/usr/ted/example1) COMPONENT('THEPRODUCT')
Sorting by the 'Event Time' column will put all the data into chronological order. To do so:

The following is the same Timed Event Data/IPST as above but in a form that can be imported into a spreadsheet program. To do so:

- Edit this data set (or a copy) and remove everything above the spreadsheet data (including these directions.)
- When you download or FTP the data set, specify ASCII or TEXT.
- Do NOT download as Binary.
- When importing into the spreadsheet program, indicate that the data is delimited with a delimiter character of: ;
- Sorting by the 'Event Time' column will put all the data into chronological order.
Sorting by the 'Event Thread' and the 'Event Time' columns will group all the related events in chronological order. Examining the 'Thread Prior Event Delta' will help identify events that took a long time.

The formatted Timed Event Data Report will consist of:

- A header section for the start of the report.
Timed Event Data Table sections with each Timed Event Data Table section comprised of a header section, the Timed Event Data Table formatted information, Timed Event Data Table formatted entries, and a trailer section.

A header section for the IPST (IPL Statistics Table) followed by the formatted IPST.

The spreadsheet format of the IPST and Timed Event Data entries.

The following describes each of the formatted sections in more detail:

The header section for the start of the report contains the text "IBM z/OS Timed Event Data Report" and the following fields:

- **Level**: The product and version of the Timed Event Data Report REXX exec (IEAVFTED).
- **Date/Time of Report**: The local date and time when the report was run.
- **Component Filter**: The filter used to select which entries to format. If no filter was specified, ALL is displayed.
- **Sysplex**: The name of the sysplex for the system from which the report was obtained.
- **System**: Name of the system from which the report was obtained.
- **FMID**: The FMID of the system from which the report was obtained and the z/OS release level.
- **Machine**: The model of the machine where the report was run.
- **Online CPs**: The number of online standard CPs, IBM zEnterprise® Application Assist Processors (zAAPs) and IBM z Integrated Information Processors (zIIPs).
- **IPL Start Date/Time**: The local date and time when the IPL was started for the system from which the report was obtained.
- **Total Timed Event Data Table Storage**: The total number of hexadecimal bytes of storage that is currently in use for the Timed Event Data Tables.

The header section for a Timed Event Data Table includes the following fields:

- **Component**: The value specified for the COMPNAME keyword on the IEATEDS REGISTER request.
- **Address**: The address in storage where the Timed Event Data Table resides.
- **Table Size**: Number of hexadecimal bytes allocated for the Timed Event Data Table.
- **Register Date/Time**: The local date and time when the Timed Event Data Table was registered with the IEATEDS REGISTER request.
- **Requested MaxEvents**: The maximum number of events originally specified on the IEATEDS REGISTER request.
- **Resultant MaxEvents**: The maximum number of events for this Timed Event Data Table that can be recorded with the IEATEDS RECORD request. This value may be the requested value, or a reduced value that allowed the Timed Event Data Table to be built within the 2M-byte storage limit.
- **Current® NumEvents**: The number of events that have been recorded thus far.
- **Overflow NumEvents**: The number of events that were not recorded because the Timed Event Data Table is full.

Following the Timed Event Data Table formatted header section, the report continues with zero or more formatted events that were recorded with IEATEDS RECORD requests, with each request having the following fields:
- **EntryNum**: This will start with 1 for each Timed Event Data Table and will increment for each event.
- **Event Type/Thread**: The type will be Start, Mid, or End as was specified for the EVENTTYPE keyword on the IEATEDS RECORD request. For Timed Event Data entries, the Thread, which is the value specified on the EVENTTREAD keyword of the IEATEDS RECORD request, will follow as formatted hex and again as formatted EBCDIC contained within asterisk borders. For IPST entries, the thread will always be in EBCDIC.
- **Event Date/Time**: The local date and time that the event was recorded.
- **"*** Incomplete Event ***"** will be displayed if it is determined that the entry is incomplete. In this case, the data for this event should be ignored.
- **Description**: The value specified for the DESCRIPTION keyword on the IEATEDS RECORD request.
- **HASN**: The ASID (address space identity) for the home address space at the time of IEATEDS RECORD request.
- **PASN**: The ASID (address space identity) for the primary address space at the time of IEATEDS RECORD request.
- **Jobname**: The jobname for the home address space at the time of the IEATEDS RECORD request.
- **TCB**: The TCB (Task Control Block) address at the time of IEATEDS RECORD request (which will be zero when running as an SRB).
- **Module/Level/Offset**: The values specified on the MODNAME and MODLEVEL keywords on the IEATEDS RECORD request, and the offset in the module where the IEATEDS RECORD request was issued. Note that the offset is calculated by obtaining the difference between the current location and SYSECT which names the current control section but which might not necessarily be the name of the module.
- **SRB/Task Time**: The SRB and task time values for the home address space at the time of IEATEDS RECORD request.
- **User Data**: The value(s) specified for the USERDATA keyword on the IEATEDS RECORD request, displayed as both printable hex and printable EBCDIC enclosed in asterisk borders. Note that zeros are appended to the user data to ensure that there are 16 bytes of data in the case where the specified user data combined size is less than 16 bytes.
- **OUXBFCON**: The formatted time from OUXBFCON which contains the accumulated I/O FICON® connect time for the address space.
- **OUXBFDIS**: The formatted time from OUXBFDIS which contains the accumulated I/O FICON disconnect time for the address space.
- **OUXBFMNO**: The formatted time from OUXBFMNO which contains the FICON magic number - for every I/O interrupt from a device attached to a FICON native CHPID, IOS will add one millisecond to this field.
- **OUXBFWAIT**: The formatted time from OUXBFWAIT which contains the accumulated I/O FICON wait time for the address space. This value includes pending time and control unit queue time.
- **IPL Start Delta**: The elapsed time from the start of the IPL to the time of the IEATEDS RECORD request for this event.
- **Timed Event Data Registration Delta**: The elapsed time from when the IEATEDS REGISTER request was made to the time of the IEATEDS RECORD request for this event.
Thread Start Event Delta: The elapsed time from when the IEATEDS RECORD request with an EVENTTYPE of START was made to the time of the IEATEDS RECORD request for this event and this event has an EVENTTHREAD value that matches the START event.

Thread Prior Event Delta: The elapsed time from when the IEATEDS RECORD request for the prior event in the same Timed Event Data Table with the same Event Thread was made to the time of the IEATEDS RECORD request for this event.

The trailer section for the Timed Event Data Table includes the following fields:

- Component: The value specified for the COMPONENT keyword on the IEATEDS REGISTER request.
- Number of Events: The number of processed START, MID, and END events. The number of start events and end events should normally match unless there were incomplete entries or an overflow of entries.

Note that error messages may be issued if a storage access error occurs, in which case the IPST, one of more Timed Event Data Tables, or Timed Event Data Table entries may be missing from the report.

Other error messages may be issued for data set errors or processing errors.

The Timed Event Data Report REXX exec provides the following return codes:

- Return Code=d'00' - Report written successfully.
- Return Code=d'16' - Report was not completed. An error message will be output to either the screen or within the report.

If you want to load the Timed Event Data Report into a spreadsheet program, perform the following:

- If the Timed Event Data Report was placed into a 512 character data set or a z/OS UNIX file, edit the data set (or a copy) to delete everything above the spreadsheet data (including the directions).
- If the Timed Event Data Report was placed into a 255 character data set (i.e., obtained from a dump under IPCS), invoke IEAVFTED with the IPCSDA option which will extract the spreadsheet data from the 255 character data set and place it into either a pre-allocated 512 character data set or a z/OS UNIX file.
- When you download or FTP the data set, choose the download options of ASCII or TEXT. Do not download as binary.
- When importing into the spreadsheet program, indicate that the data is delimited with a delimiter of a semicolon (or the character that was specified with the SS keyword when IEAVFTED was invoked to generate the Timed Event Data Report).
- Sorting by the 'Event Time' column will put all the data into chronological order.
- Sorting by the 'Event Thread' and the 'Event Time' columns will group all the related events in chronological order. Examining the 'Thread Prior Event Delta' will help identify events that took a long time.
Chapter 47. IEATXDC — Transactional execution diagnostic controls

Description
In support of the diagnostic controls of transactional execution, as defined in the z/Architecture Principles of Operation, the following services are provided:

For the current task,
• Indicate the scope of the diagnostic controls.
• Set the diagnostic controls for "no abort".
• Set the diagnostic controls for "abort every".
• Set the diagnostic controls for "abort random".

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state with PSW key 8-15. SCOPE=ALL requires supervisor state; if a problem state caller indicates SCOPE=ALL, it is treated as SCOPE=PROBLEM.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller may hold any lock(s). No locks are required.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>None.</td>
</tr>
</tbody>
</table>

Programming requirements
None.

Restrictions
None.

Input register information
Before issuing the IEATXDC macro, the caller does not have to place any information into any general purpose register (GPR).

Before issuing the IEATXDC macro, the caller does not have to place any information into any access register (AR).

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
IEATXDC macro

2-13 Unchanged
14 Used as a work register by the system
15 Return code

When control returns to the caller, the ARs contain:

Register

Contents

0-1 Used as work registers by the system
2-13 Unchanged
14-15 Used as work registers by the system

Performance implications
None.

Syntax
The IEATXDC macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEATXDC.</td>
</tr>
<tr>
<td>IEATXDC</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEATXDC.</td>
</tr>
<tr>
<td>SCOPE=PROBLEM</td>
<td></td>
</tr>
<tr>
<td>SCOPE=ALL</td>
<td></td>
</tr>
<tr>
<td>,OPERATION=NO_ABORT</td>
<td></td>
</tr>
<tr>
<td>,OPERATION=SET_EVERY</td>
<td></td>
</tr>
<tr>
<td>,OPERATION=SET_RANDOM</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12) or (15), (GPR15), (REG15), or (R15).</td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEATXDC macro invocation. The name must conform to the rules for an ordinary assembler language symbol.
IEATXDC macro

SCOPE=PROBLEM
SCOPE=ALL
A required parameter that identifies the scope of the diagnostic controls.

SCOPE=PROBLEM
indicates that the diagnostic controls apply only for problem state
transactional execution.

SCOPE=ALL
indicates that the diagnostic controls apply both to problem state and
supervisor state transactional execution. If a problem state caller requests
SCOPE=ALL, however, it is treated as SCOPE=PROBLEM.

,OPERATION=NO_ABORT
,OPERATION=SET_EVERY
,OPERATION=SET_RANDOM
A required parameter that identifies the type of operation to perform.

,OPERATION=NO_ABORT
indicates to set the transactional diagnostic controls for this task so that the
system will not apply its SET_EVERY or SET_RANDOM rules.
Transactions themselves may still abort for all the defined architectural
reasons.

,OPERATION=SET_EVERY
indicates to set the transactional diagnostic controls for this task to request
abort of every nonconstrained transaction.

,OPERATION=SET_RANDOM
indicates to set the transactional diagnostic controls for this task to request
abort of random nonconstrained transactions.

,RETCODE=retcode
An optional output parameter into which the return code is to be copied from
GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without
parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or
(15), (GPR15), (REG15), or (R15).

ABEND codes
None.

Return codes
When the IEATXDC macro returns control to your program, GPR 15 (and retcode,
when you code RETCODE) contains a return code.

The following table identifies the hexadecimal return and reason codes.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0           | Meaning: Successful completion. Diagnostic controls are set to the requested value
|             | Action: None required. |
| 4           | Meaning: Warning. The machine does not support transactional execution. Diagnostic
|             | controls are not set. |
|             | Action: Avoid calling IEATXDC when the machine does not support transactional
|             | execution. |
### Table 54. Return codes for the IEATXDC Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 8           | **Meaning:** Unexpected input.  
|             | **Action:** Check for possible storage overlay. |
| 12          | **Meaning:** Service called in SRB mode.  
|             | **Action:** Avoid using IEATXDC in SRB mode. |

#### Examples

None.
Chapter 48. IEAVAPE — Allocate_Pause_Element

Description

Allocate_Pause_Element obtains a pause element token (PET), which uniquely identifies a pause element. The PET is used as input to the following services:

- Pause
- Release
- Transfer
- Deallocate_Pause_Element

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td></td>
</tr>
<tr>
<td>• For auth_level=IEA_UNAUTHORIZED: Task</td>
<td></td>
</tr>
<tr>
<td>• For auth_level=IEA_AUTHORIZED: Task or SRB</td>
<td></td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td></td>
</tr>
<tr>
<td>• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN</td>
<td></td>
</tr>
<tr>
<td>• For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN</td>
<td></td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td></td>
</tr>
<tr>
<td>• When supervisor state and PSW key 0: The local lock may be held.</td>
<td></td>
</tr>
<tr>
<td>• When problem state, or not PSW key 0: No locks may be held.</td>
<td></td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.
IEAVAPE callable service

**Input register information**
Before calling Allocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 72-byte register save area.</td>
</tr>
</tbody>
</table>

**Output register information**
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return Code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**
None.

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEAVAPE</td>
<td>(return_code, auth_level, pause_element_token)</td>
</tr>
</tbody>
</table>

**Parameters**
The parameters are explained as follows:

- **return_code**
  - Returned parameter
  - Type: Integer
IEAVAPE callable service

- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Allocate_Pause_Element service.

`auth_level`
Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Represents one or more possible levels of the pause element being allocated. The calling program can use the constants defined in IEAASM or IEAC, as appropriate. The level desired results from adding the values of the required types together. The authorization type is not optional.

For instance, the level to allocate authorized pause elements that are checkpoint/restart tolerant is IEA_AUTHORIZED + IEA_CHECKPOINTOK, or 3.

The following levels are supported:

<table>
<thead>
<tr>
<th>IEAASM and IEAC defined constants</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>When using the allocated pause element through other services, either auth_level IEA_UNAUTHORIZED or IEA_AUTHORIZED can be used.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>When using the allocated pause element through other services, auth_level=IEA_AUTHORIZED will be required. Caller must be both key 0 and supervisor state.</td>
</tr>
</tbody>
</table>

Table 56. Checkpoint/Restart Toleration - only available when the CVTPAUS4 bit is set in the CVT.

<table>
<thead>
<tr>
<th>IEAASM and IEAC defined constants</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_CHECKPOINTOK</td>
<td>2</td>
<td>The application can tolerate the pause elements’ not being restored upon a restart after a checkpoint.</td>
</tr>
</tbody>
</table>

Note: If the IEA_CHECKPOINTOK value is not added to the authorization value, checkpoints cannot be taken when an allocated pause element exists.

`pause_element_token`
Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element which you can use to synchronize the processing of a task.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.
### IEAVAPE callable service

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decimal (Hex)</strong></td>
<td><strong>Equate symbol</strong></td>
</tr>
<tr>
<td>00 (0) IEA_SUCCESS</td>
<td><strong>Meaning:</strong> Successful completion. <strong>Action:</strong> None.</td>
</tr>
<tr>
<td>24 (18) IEA_LOCK_HELD</td>
<td><strong>Meaning:</strong> Program error. If the auth_level indicates AUTHORIZED, locks other than the local lock are held. If the auth_level indicates UNAUTHORIZED, locks are held. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>36 (24) IEA_UNSUPPORTED_MVS_RELEASE</td>
<td><strong>Meaning:</strong> Environmental error. The system release does not support this service. The system rejects the service call. <strong>Action:</strong> Run the program on a system that supports the service.</td>
</tr>
<tr>
<td>40 (28) IEA_PE_NOT_HOME</td>
<td><strong>Meaning:</strong> Program error. The auth_level value specified in the call is not valid. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>44 (2C) IEA_XFER_TO_SELF</td>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>48 (30) IEA_XFER_FAILED</td>
<td><strong>Meaning:</strong> Environmental error. The system could not obtain storage for a pause element. The system rejects the service call. <strong>Action:</strong> Retry the request later. If the problem persists, consult your system programmer.</td>
</tr>
<tr>
<td>56 (38) IEA_NO_PETS_AVAILABLE</td>
<td><strong>Meaning:</strong> There are no pause element tokens available. <strong>Action:</strong> Retry the request later.</td>
</tr>
<tr>
<td>4095 (FFF) IEA_UNEXPECTED_ERROR</td>
<td><strong>Meaning:</strong> This service routine encountered an unexpected error. The system rejects this service request. <strong>Action:</strong> Contact IBM support.</td>
</tr>
</tbody>
</table>
Chapter 49. IEAVAPE2 — Allocate_Pause_Element

Description
Allocate_Pause_Element obtains a pause element token (PET), which uniquely identifies a pause element. The PET is used as input to the following services:

- Pause
- Release
- Transfer
- Deallocate_Pause_Element

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td></td>
</tr>
<tr>
<td>For LINKAGE=BRANCH, supervisor state and PSW key 0.</td>
<td></td>
</tr>
<tr>
<td>For LINKAGE=SVC:</td>
<td></td>
</tr>
<tr>
<td>- Working with an IEAAUTHORIZED pause element, supervisor state and PSW key 0-7.</td>
<td></td>
</tr>
<tr>
<td>- Working with an IEAUNAUTHORIZED pause element, problem state and any PSW key.</td>
<td></td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td></td>
</tr>
<tr>
<td>For LINKAGE=BRANCH: Task or SRB</td>
<td></td>
</tr>
<tr>
<td>For LINKAGE=SVC: Task</td>
<td></td>
</tr>
<tr>
<td>Cross memory mode</td>
<td></td>
</tr>
<tr>
<td>For LINKAGE=BRANCH: Any PASN, any HASN, any SASN</td>
<td></td>
</tr>
<tr>
<td>For LINKAGE=SVC: PASN=HASN=SASN</td>
<td></td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
</tbody>
</table>

Interrupt status: Enabled for I/O and external interrupts.

Locks:
- For LINKAGE=BRANCH: The local lock may be held.
- For LINKAGE=SVC: No locks may be held.

Control parameters: Must be in the primary address space and addressable by the caller.

Programming requirements
Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>
IEAVPE2 callable service

Restrictions
Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Allocate_Pause_Element cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Key 1-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC and pause_element_owner_stoken as binary zero.

Input register information
Before calling Allocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 72-byte register save area.</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.
IEAVAPE2 callable service

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEAVAPE2</td>
<td>(,)return_code,(,)pause_element_auth_level,(,)pause_element_token,(,)pause_element_owner_stoken,(,)owner_termination_release_code,(,)linkage)</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

\textbf{return\_code}

- Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Allocate_Pause_Element service.

\textbf{pause\_element\_auth\_level}

- Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Represents one or more possible levels of the pause element being allocated. The calling program can use the constants defined in IEAASM or IEAC, as appropriate. The level desired results from adding the values of the required types together. The authorization type is not optional.

For instance, the level to allocate authorized pause elements that are checkpoint/restart tolerant is \(\text{IEA\_AUTHORIZED + IEA\_CHECKPOINTOK}\), or 3.

The following levels are supported:

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
\textbf{IEAASM and IEAC defined constants} & \textbf{Value (hexadecimal)} & \textbf{Meaning} \\
\hline
\text{IEA\_UNAUTHORIZED} & 0 & When using the allocated pause element through other services, either \text{pause\_element\_auth\_level} \text{IEA\_UNAUTHORIZED} or \text{IEA\_AUTHORIZED} can be used. \\
\hline
\text{IEA\_AUTHORIZED} & 1 & When using the allocated pause element through other services, \text{pause\_element\_auth\_level} \text{IEA\_AUTHORIZED} is required. Caller must be both key 0 and supervisor state. \\
\hline
\end{tabular}
\caption{Authorization}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
\textbf{IEAASM and IEAC defined constants} & \textbf{Value (hexadecimal)} & \textbf{ Meaning} \\
\hline
\text{IEA\_CHECKPOINTOK} & 2 & The application can tolerate the pause elements' not being restored upon a restart after a checkpoint. \\
\hline
\end{tabular}
\caption{Checkpoint/Restart Toleration - only available when the CVTPAUS4 bit is set in the CVT.}
\end{table}
IEAVAPE2 callable service

**Note:** If the IEA_CHECKPOINTOK value is not added to the authorization value, checkpoints cannot be taken when an allocated pause element exists.

**pause_element_token**

Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies a pause element which you can use to synchronize the processing of a task or SRB.

**pause_element_owner_stoken**

Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 8 bytes

Specifies the space token (STOKEN) of the address space which is to be considered the owner of the Pause Element being allocated. Specify one of the following values:
- Binary zero: indicate the system should make the current primary address space the owner of the Pause Element. This is the only value valid for key 8-15 problem state callers.
- A valid STOKEN, indicate the system should make the address space with the matching STOKEN the owner for the pause element.

When the CMRO task (the first job step task) of an address space terminates, the system will release and deallocate any pause elements owned by the CMRO task's home address space. The table below describes exactly when the system will release and/or deallocate a Pause Element:

<table>
<thead>
<tr>
<th>Allocation Service version:</th>
<th>Deallocation Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAVAPE</td>
<td>The PE will be deallocated by the system when one of the following events occurs:</td>
</tr>
<tr>
<td></td>
<td>• The PE was never used to pause a task or SRB and the CMRO task for the space which allocated it terminates.</td>
</tr>
<tr>
<td></td>
<td>• The PE is being used to pause a task or SRB which is asynchronously terminated via CALLRTM TYPE=ABTERM (for example, cancel or detach) or a PURGEDQ</td>
</tr>
<tr>
<td></td>
<td>• The CMRO task of the home address space of the task or SRB which last used the PE terminates and the PE is not being used to pause an SRB.</td>
</tr>
<tr>
<td></td>
<td>The home address space of the task or SRB which last used the PE terminates.</td>
</tr>
</tbody>
</table>
IEAVAPE2 callable service

<table>
<thead>
<tr>
<th>Allocation Service version:</th>
<th>Deallocation Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAVAPE2</td>
<td>The PE will be deallocated by the system when one of the following events occurs:</td>
</tr>
<tr>
<td></td>
<td>- The CMRO task of the address space specified by pause_element_owner_stoken terminates. If the PE is being used to pause a DU when the CMRO task terminates, the system will release the DU using the owner_termination_release_code before the PE is deallocated. Note that in this case, the UPET returned will be 16 bytes of binary zeros, an invalid value.</td>
</tr>
<tr>
<td></td>
<td>- The PE is being used to pause a task or SRB which is asynchronously terminated via CALLRTM TYPE=ABTERM (for example, cancel or detach) or a PURGEDQ</td>
</tr>
<tr>
<td></td>
<td>- The PE is being used to pause a task or SRB when the home address space of the task or SRB is terminated</td>
</tr>
<tr>
<td></td>
<td>- The CMRO task of the address space which owns the PE terminates and the PE is not being used to pause an SRB.</td>
</tr>
</tbody>
</table>

The address space which owns the PE terminates. Note: A PE is considered as "being used to pause a task or SRB," when the PE is not Reset or Prereleased.

\*owner_termination_release_code\*

Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Specifies the release code which will be returned to a paused DU if the system deallocates the pause element while it is being used to pause a task or SRB, due to the CMRO task of its owning address space terminating.

**Note:** If the system deallocates a PE due to its owner terminating while the PE was not being used to pause a task or SRB, future attempts to use the PE will fail with a return code indicating the PETOKEN was stale or the PE is in an invalid state.

\*linkage\*

Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Allocate_Pause_Element service routine is to be invoked. The following options are supported:
**IEAVAPE2 callable service**

*Table 59. Linkage option*

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_SVC</td>
<td>0</td>
<td>The Allocate_Pause_Element service routine will be invoked via an SVC linkage. This option can be used when in non-cross memory task mode, any key, and either problem state or supervisor state.</td>
</tr>
<tr>
<td>IEA_LINKAGE_BRANCH</td>
<td>1</td>
<td>The Allocate_Pause_Element service routine will be invoked via a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.</td>
</tr>
</tbody>
</table>

**ABEND codes**

None.

**Return codes**

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (0)</td>
<td>IEA_SUCCESS</td>
<td><strong>Meaning:</strong> Successful completion. <strong>Action:</strong> None.</td>
</tr>
<tr>
<td>24 (18)</td>
<td>IEA_LOCK_HELD</td>
<td><strong>Meaning:</strong> Program error. One or more locks other than the local lock are held. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>IEA_UNSUPPORTED_MVS_RELEASE</td>
<td><strong>Meaning:</strong> Environmental error. The system release does not support this service. The system rejects the service call. <strong>Action:</strong> Run the program on a system that supports the service.</td>
</tr>
<tr>
<td>40 (28)</td>
<td>IEA_INVALID_AUTHCODE</td>
<td><strong>Meaning:</strong> Program error. The pause_element_auth_level value specified in the call is not valid. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>IEA_INVALID_MODE</td>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
</tbody>
</table>
## IEAVAPE2 callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 48 (30)                     | IEA_OUT_OF_STORAGE               | **Meaning:** Environmental error. The system could not obtain storage for a pause element. The system rejects the service call.  
**Action:** Retry the request later. If the problem persists, consult your system programmer. |
| 56 (38)                     | IEA_NO_PETS_AVAILABLE            | **Meaning:** There are no pause element tokens available.  
**Action:** Retry the request later. |
| 4095 (FFF)                  | IEA_UNEXPECTED_ERROR             | **Meaning:** This service routine encountered an unexpected error. The system rejects this service request.  
**Action:** Contact IBM support. |
| 84 (54)                     | IEA_INVALID_LINKAGE              | **Meaning:** Program error. The linkage value specified is not valid. The system rejects the service call  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 88 (58)                     | IEA_INVALID_OWNER_STOKEN         | **Meaning:** Program error. The stoken specified for pause_element_owner_stoken is not valid.  
**Action:** Obtain the correct stoken of the target and reissue the call |
| 96 (60)                     | IEA_UNAUTH_NONZERO_OWNER_STOKEN  | **Meaning:** Program error. A key 8-15 problem state caller specified a nonzero value for pause_element_owner_stoken  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 100 (64)                    | IEA_INVALID_AUTHLEVEL_AUTHCODE   | **Meaning:** The pause_element_auth_level value specified in the call is not valid. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
IEAVAPE2 callable service
Chapter 50. IEAVDPE — Deallocate_Pause_Element

Description

Deallocate_Pause_Element frees a pause element that is no longer needed.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>For auth_level=IEA_UNAUTHORIZED: Task</td>
</tr>
<tr>
<td></td>
<td>For auth_level=IEA_AUTHORIZED: Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN</td>
</tr>
<tr>
<td></td>
<td>For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit addressing mode.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>When supervisor state and PSW key 0: The local lock may be held.</td>
</tr>
<tr>
<td></td>
<td>When problem state, or not PSW key 0: No locks may be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling Deallocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
</table>
IEAVDPE callable service

1 Address of the parameter address list.
13 Address of a 72-byte register save area.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEAVDPE</td>
<td>&quot;(return_code,auth_level,pause_element_token)&quot;</td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained as follows:

return_code
 Returned parameter
• Type: Integer
• Character Set: N/A
• Length: 4 bytes

Contains the return code from the Deallocate_Pause_Element service.
IEAVDPE callable service

auth_level
Supplied parameter
• Type: Integer
• Character Set: N/A
• Length: 4 bytes

Indicates the maximum authorization level of the pause element being deallocated. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (HEX)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>This pause element being deallocated must have been allocated with auth_level=IEA_UNAUTHORIZED.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>This pause element being deallocated must have been allocated with auth_level=IEA_AUTHORIZED.</td>
</tr>
</tbody>
</table>

pause_element_token
Supplied parameter
• Type: Character string
• Character Set: N/A
• Length: 16 bytes

Contains the pause element token that identifies the pause element that is no longer needed.

ABEND codes
None.

Return codes
When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
</tr>
<tr>
<td>Equate symbol</td>
<td></td>
</tr>
</tbody>
</table>

| 00 (00)         | Meaning: Successful completion |
|                 | Action: None.                 |
| IEA_SUCCESS     |                                   |
| 04 (04)         | Meaning: Program error. The specified pause element token is not valid. The system rejects the service call. |
|                 | Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| 08 (08)         | Meaning: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. |
|                 | Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| 24 (18)         | Meaning: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. |
|                 | Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
## IEAVDPE callable service

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
</tr>
<tr>
<td>Equate symbol</td>
<td></td>
</tr>
<tr>
<td>32 (20)</td>
<td><strong>IEA_PE_BAD_STATE</strong>&lt;br&gt;&lt;br&gt;&lt;strong&gt;Meaning**: Program error. The pause element associated with the specified pause element token is not valid or has already been paused. A paused PE must be released before it is deallocated. This return code also can indicate that the address space associated with the pause element is ending or has ended and that the system freed the pause element.&lt;br&gt;&lt;br&gt;&lt;strong&gt;Action**: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>36 (24)</td>
<td><strong>IEA_UNSUPPORTED_MVS_RELEASE</strong>&lt;br&gt;&lt;br&gt;&lt;strong&gt;Meaning**: Environmental error. The system release does not support this service. The system rejects the service call.&lt;br&gt;&lt;br&gt;&lt;strong&gt;Action**: Run the program on a system that supports the service.</td>
</tr>
<tr>
<td>40 (28)</td>
<td><strong>IEA_INVALID_AUTHCODE</strong>&lt;br&gt;&lt;br&gt;&lt;strong&gt;Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call.&lt;br&gt;&lt;br&gt;&lt;strong&gt;Action**: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td><strong>IEA_INVALID_MODE</strong>&lt;br&gt;&lt;br&gt;&lt;strong&gt;Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.&lt;br&gt;&lt;br&gt;&lt;strong&gt;Action**: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>60 (3C)</td>
<td><strong>IEA_AUTH_TOKEN</strong>&lt;br&gt;&lt;br&gt;&lt;strong&gt;Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call.&lt;br&gt;&lt;br&gt;&lt;strong&gt;Action**: Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td>64 (40)</td>
<td><strong>IEA_PE_NOT_HOME</strong>&lt;br&gt;&lt;br&gt;&lt;strong&gt;Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address.&lt;br&gt;&lt;br&gt;&lt;strong&gt;Action**: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>4095 (FFF)</td>
<td><strong>IEA_UNEXPECTED_ERROR</strong>&lt;br&gt;&lt;br&gt;&lt;strong&gt;Meaning**: This service routine encountered an unexpected error. The system rejects this service request.&lt;br&gt;&lt;br&gt;&lt;strong&gt;Action**: Contact IBM support.</td>
</tr>
</tbody>
</table>
Chapter 51. IEAVDPE2 — Deallocate_Pause_Element

Description
Deallocate_Pause_Element frees a pause element that is no longer needed.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>For LINKAGE=BRANCH, supervisor state and PSW key 0.</td>
</tr>
<tr>
<td></td>
<td>For LINKAGE=SVC:</td>
</tr>
<tr>
<td></td>
<td>– Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7.</td>
</tr>
<tr>
<td></td>
<td>– Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>For LINKAGE=BRANCH: Task or SRB</td>
</tr>
<tr>
<td></td>
<td>For LINKAGE=SVC: Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>For LINKAGE=BRANCH: Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td></td>
<td>For LINKAGE=SVC: PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit addressing mode.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>For LINKAGE=BRANCH: The local lock may be held.</td>
</tr>
<tr>
<td></td>
<td>For LINKAGE=SVC: No locks may be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements
Either link the object code of the calling program with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions
Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Key 1-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC.
IEAVDPE2 callable service

Input register information
Before calling Deallocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 72-byte register save area.</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEAVDPE2</td>
<td>_return_code&lt;br&gt;_pause_element_token&lt;br&gt;_linkage</td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained as follows:
IEAVDPE2 callable service

**return_code**
Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Deallocate_Pause_Element service.

**pause_element_token**
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element that is no longer needed.

**linkage**
Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Deallocate_Pause_Element service routine is to be invoked. The following options are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_SVC</td>
<td>0</td>
<td>The Deallocate_Pause_Element service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and in either problem state or supervisor state.</td>
</tr>
<tr>
<td>IEA_LINKAGE_BRANCH</td>
<td>1</td>
<td>The Deallocate_Pause_Element service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.</td>
</tr>
</tbody>
</table>

**ABEND codes**
None.

**Return codes**
When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00 (00)                       | IEA_SUCCESS   | **Meaning:** Successful completion  
**Action:** None. |
### IEAVDPE2 callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04 (04)                       | IEA_PE_TOKEN_BAD               | **Meaning:** Program error. The specified pause element token is not valid. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 08 (08)                       | IEA_PE_TOKEN_STALE             | **Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 24 (18)                       | IEA_LOCK_HELD                  | **Meaning:** Program error. One or more locks other than the local lock are held. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 32 (20)                       | IEA_PE_BAD_STATE               | **Meaning:** Program error. The pause element associated with the specified pause element token is invalid or has already been paused. A paused PE must be released before it is deallocated. This return code also can indicate that the address space associated with the pause element is ending or has ended and that the system freed the pause element.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 36 (24)                       | IEA_UNSUPPORTED_MVS_RELEASE     | **Meaning:** Environmental error. The system release does not support this service. The system rejects the service call.  
**Action:** Run the program on a system that supports the service. |
| 44 (2C)                       | IEA_INVALID_MODE               | **Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 64 (40)                       | IEA_PE_NOT_HOME                | **Meaning:** Program error. The pause element token was for an unauthorized pause element allocated to another address space.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
### IEAVDPE2 callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 84 (54)                      | IEA_INVALID_LINKAGE         | **Meaning:** Program error. The linkage value specified is not valid. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 4095 (FFF)                   | IEA_UNEXPECTED_ERROR        | **Meaning:** This service routine encountered an unexpected error. The system rejects this service request.  
**Action:** Contact IBM support. |
IEAVDPE2 callable service
Chapter 52. IEAVPSE — Pause service

Description

Call Pause to make the current task nondispatchable. Once you pause a task, it remains nondispatchable until a Release service specifying the same PET is called. That is, the program issuing the Pause does not receive control back until after the Release occurs.

If a Release service specifying the same PET is called before Pause, the system returns control immediately to the calling program, and the task is not paused.

When you use Pause, it returns an updated PET; you use this updated PET to either deallocate or reuse the PE.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>• When supervisor state and PSW key 0: Task or SRB.</td>
</tr>
<tr>
<td></td>
<td>• When problem state, or not PSW key 0: Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>• For auth_level=IEA_UNAUTHORIZED:</td>
</tr>
<tr>
<td></td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td></td>
<td>• For auth_level=IEAAUTHORIZED:</td>
</tr>
<tr>
<td></td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit addressing mode.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the calling program’s object code with the linkable stub routine (IEACSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

When the calling program is running auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only pause another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.
Input register information
Before calling the Pause service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 72-byte register save area.</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEAVPSE</td>
<td>i(return_code, auth_level, pause_element_token, updated_pause_element_token, release_code)</td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained as follows:

return_code
Returned parameter
IEAVPSE callable service

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Pause service.

\texttt{auth\_level}

Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum level that the specified pause element was allocated with. IEAASM and IEAC define constants IEA\_UNAUTHORIZED and IEA\_AUTHORIZED, which the calling program can use. The following levels are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (HEX)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>The pause element being paused must have been allocated with auth_level=IEA_UNAUTHORIZED.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>The pause element being paused must have been allocated with auth_level=IEA_AUTHORIZED.</td>
</tr>
</tbody>
</table>

\texttt{pause\_element\_token}

Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element being used to pause the current task. You obtain the PET from the Allocate_Pause_Element service.

Once you use a PET in a call to the Pause service, you cannot reuse the PET on a second call to Pause or on a call to Transfer. The Pause service returns a new PET in updated\_pause\_element\_token. The new PET now identifies the pause element used to Pause the task; use the new PET the next time you make a Pause request using the same Pause element.

\texttt{updated\_pause\_element\_token}

Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A new pause element token that identifies the pause element originally identified by the PET specified in pause\_element\_token, which cannot be reused after a successful call to Pause.

\texttt{release\_code}

Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code, specified by the issuer of the Release service. A Release that specified this code released the task from its paused condition.
### ABEND codes

<table>
<thead>
<tr>
<th>Abend Code</th>
<th>Reason Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC7</td>
<td>001A0001</td>
<td>This is an internal error. Contact IBM support.</td>
</tr>
</tbody>
</table>

### Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decimal (Hex)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Equate symbol</strong></td>
<td></td>
</tr>
<tr>
<td>00 (00)</td>
<td><strong>IEA_SUCCESS</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Successful completion.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: None</td>
</tr>
<tr>
<td>04 (04)</td>
<td><strong>IEA_PE_TOKEN_STALE</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>08 (08)</td>
<td><strong>IEA_PE_TOKEN_STALE</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET be returned on Pause or Transfer.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>12 (0C)</td>
<td><strong>IEA_DUPLICATE_PAUSE</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: The work unit has already been paused using the specified pause element token. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>24 (18)</td>
<td><strong>IEA_LOCK_HELD</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>32 (20)</td>
<td><strong>IEA_PE_BAD_STATE</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it.</td>
</tr>
<tr>
<td>36 (24)</td>
<td><strong>IEA_UNSUPPORTED_MVS_RELEASE</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Environmental error. The system release does not support this service. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Run the program on a system that supports the service.</td>
</tr>
<tr>
<td>40 (28)</td>
<td><strong>IEA_INVALID_AUTHCODE</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Program error. The auth_level value specified in the call is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
</tbody>
</table>
### IEAVPSE callable service

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
</tr>
<tr>
<td>Equate symbol</td>
<td></td>
</tr>
</tbody>
</table>
| 44 (2C) IEA_INVALID_MODE | **Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 52 (34) IEA_ALREADY_SUSPENDED | **Meaning:** The pause element was already paused.  
**Action:** Check the calling program for a probable coding error and correct the program and rerun it. |
| 60 (3C) IEA_AUTH_TOKEN | **Meaning:** Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call.  
**Action:** Program error. The specified pause element token is not valid. The system rejects the service call. |
| 64 (40) IEA_PE_NOT_HOME | **Meaning:** Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 4095 (FFF) IEA_UNEXPECTED_ERROR | **Meaning:** This service routine encountered an unexpected error. The system rejects this service request.  
**Action:** Contact IBM support. |
IEAVPSE callable service
Chapter 53. IEAVPSE2 — Pause service

Description

Call Pause to make the current task or SRB nondispatchable. When you pause a task or SRB, it remains nondispatchable until a Release or Transfer specifying the same PET is called. That is, the program issuing the Pause does not receive control back until after the Release or Transfer occurs. At that time, the returned release_code will contain a value supplied by the associated Release or Transfer request.

If a Release service specifying the same PET is called before Pause, the system returns control immediately to the calling program, and the task or SRB is not paused.

When you use Pause, it returns an updated PET; you use this updated PET to either deallocate or reuse the PE.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
</table>
| Minimum authorization: | • For LINKAGE=BRANCH, supervisor state and PSW key 0.  
• For LINKAGE=SVC:  
  – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7.  
  – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key. |
| Dispatchable unit mode: | • For LINKAGE=BRANCH: Task or SRB  
• For LINKAGE=SVC: Task |
| Cross memory mode: | • For LINKAGE=BRANCH: Any PASN, any HASN, any SASN  
• For LINKAGE=SVC: PASN=HASN=SASN |
| AMODE: | 31-bit addressing mode. |
| ASC mode: | Primary mode. |
| Interrupt status: | Enabled for I/O and external interrupts. |
| Locks: | No locks held. |
| Control parameters: | Must be in the primary address space and addressable by the caller. |

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>
IEAVPSE2 callable service

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Key 1-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC.

Pause cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information

Before calling the Pause service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 72-byte register save area.</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.
IEAVPSE2 callable service

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEAVPSE2</td>
<td></td>
</tr>
<tr>
<td>,return_code</td>
<td>Returned parameter</td>
</tr>
<tr>
<td>,pause_element_token</td>
<td>Supplied parameter</td>
</tr>
<tr>
<td>,updated_pause_element_token</td>
<td>A new pause element token that identifies the pause element being used to pause the current task or SRB. You obtain the PET from the Allocate_Pause_Element service. When you use a PET in a call to the Pause service, you cannot reuse the PET on a second call to Pause or on a call to Transfer. The Pause service returns a new PET in updated_pause_element_token. The new PET now identifies the pause element used to pause the task or SRB; use the new PET the next time you make a Pause request using the same Pause element.</td>
</tr>
<tr>
<td>,release_code</td>
<td>Returned parameter</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

**return_code**

- Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes
- Contains the return code from the Pause service.

**pause_element_token**

- Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes
- A pause element token that identifies the pause element being used to pause the current task or SRB. You obtain the PET from the Allocate_Pause_Element service.

When you use a PET in a call to the Pause service, you cannot reuse the PET on a second call to Pause or on a call to Transfer. The Pause service returns a new PET in updated_pause_element_token. The new PET now identifies the pause element used to pause the task or SRB; use the new PET the next time you make a Pause request using the same Pause element.

**updated_pause_element_token**

- Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes
- A new pause element token that identifies the pause element originally identified by the PET specified in pause_element_token. This new PET must be used in place of the PET specified in pause_element_token on future calls to the Pause, Release, Transfer, or Deallocate_Pause_Element service. If the paused workunit was released by the system (the release code is the owner_termination_release_code specified on the IEAVAPE1 allocation), the PET returned will be 16 bytes of binary zeros, an invalid value.

**release_code**

- Returned parameter
- Type: Character string
- Character Set: N/A
IEAVPSE2 callable service

- Length: 3 bytes
  The release code, specified by the issuer of the Release service. A Release that specified this code released the task or SRB from its paused condition.

**linkage**
Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Pause service routine is to be invoked. The following options are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_SVC</td>
<td>0</td>
<td>The Pause service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and either problem state or supervisor state.</td>
</tr>
<tr>
<td>IEA_LINKAGE_BRANCH</td>
<td>1</td>
<td>The Pause service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.</td>
</tr>
</tbody>
</table>

**ABEND codes**

<table>
<thead>
<tr>
<th>Abend Code</th>
<th>Reason Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC7</td>
<td>001A0001</td>
<td>This is an internal error. Contact IBM support.</td>
</tr>
</tbody>
</table>

**Return codes**

When the service returns control to your program, GPR 15 contains one of the following return codes:

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>IEA_SUCCESS</td>
<td>Meaning: Successful completion.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None</td>
</tr>
<tr>
<td>04 (04)</td>
<td>IEA_PE_TOKEN_BAD</td>
<td>Meaning: Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>IEA_PE_TOKEN_STALE</td>
<td>Meaning: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>12 (0C)</td>
<td>IEA_DUPLICATE_PAUSE</td>
<td>Meaning: The work unit has already been paused using the specified pause element token. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
</tbody>
</table>
### IEAVPSE2 callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 24 (18)                      | IEA_LOCK_HELD                | **Meaning:** Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 32 (20)                      | IEA_PE_BAD_STATE             | **Meaning:** Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it. |
| 36 (24)                      | IEA_UNSUPPORTED_MVS_RELEASE   | **Meaning:** Environmental error. The system release does not support this service. The system rejects the service call.  
**Action:** Run the program on a system that supports the service. |
| 44 (2C)                      | IEA_INVALID_MODE             | **Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 52 (34)                      | IEA_ALREADY_SUSPENDED        | **Meaning:** The pause element was already paused.  
**Action:** Check the calling program for a probable coding error and correct the program and rerun it. |
| 60 (3C)                      | IEA_AUTH_TOKEN               | **Meaning:** Program error. The caller was in Problem state or key 8, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call.  
**Action:** Program error. The specified pause element token is not valid. The system rejects the service call. |
| 64 (40)                      | IEA_PE_NOT_HOME              | **Meaning:** Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space.  
**Action:** Check the calling program for a probable coding error and correct the program and rerun it. |
| 76 (4C)                      | IEA_ABENDED_47B              | **Meaning:** After an SRB received ABEND 47B, it invoked IEAVPSE. It is not valid to invoke IEAVPSE after receiving ABEND 47B.  
**Action:** Update the calling program to not invoke IEAVPSE after ABEND 47B. |
| 80 (50)                      | IEA_IN_SUSPEND_EXIT          | **Meaning:** The suspend exit specified on SUSPEND with SPTOKEN of an SRB invoked IEAVPSE. It is not valid to invoke IEAVPSE from a suspend exit.  
**Action:** Update the calling program to not invoke IEAVPSE from a suspend exit. |
| 84 (54)                      | IEA_INVALID_LINKAGE          | **Meaning:** Program error. The linkage value specified is not valid. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
### IEAVPSE2 callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 4095 (FFF)                   | IEA_UNEXPECTED_ERROR      | **Meaning:** This service routine encountered an unexpected error. The system rejects this service request.  
**Action:** Contact IBM support. |
Chapter 54. IEAVRLS — Release

Description

Call Release to remove a task that has been paused, or to keep a task from being paused. Although a pause element can be used multiple times to pause a task, a pause element token can be used to successfully pause and release a task only once. Each time a pause element is used, the system generates a new PET to identify the pause element. The system returns the new updated PET on calls to the Pause and Transfer services.

Environment

The requirements for the caller are:

Environmental factor          Requirement
Minimum authorization:        Problem state and any PSW key.
Dispatchable unit mode:       • When supervisor state and PSW key 0: Task or SRB.
                              • When problem state, or not PSW key 0: Task.
Cross memory mode:            • For auth_level=IEA_UNAUTHORIZED:
                              PASN=HASN=SASN
                              • For auth_level=IEA_AUTHORIZED: Any PASN, any
                              HASN, any SASN
AMODE:                        31-bit addressing mode.
ASC mode:                     Primary mode.
Interrupt status:             Enabled or disabled for I/O and external interrupts.
Locks:                        May hold the CPU, local or CMS lock.
Control parameters:           Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program’s object code with the linkable stub routine (IEACSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling the Release service, the caller must ensure that the following general purpose (GPRs) contain the specified information:
IEAVRLS callable service

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>13</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
</tr>
<tr>
<td>2-13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
</tr>
<tr>
<td>2-13</td>
</tr>
<tr>
<td>14-15</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEAVRLS</td>
<td>,return_code, auth_level, target_du_pause_element_token, target_du_release_code</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

return_code

Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes
IEAVRLS callable service

Contains the return from the Release service.

auth_level
Supplied Parameter
  Type: Integer
  Character Set: N/A
  Length: 4 bytes

Indicates the maximum authorization level that the specified pause element
was allocated with. IEAASM and IEAC define constants
IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program
can use. The following levels are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (HEX)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>The pause element being released must have been allocated with auth_level=IEA_UNAUTHORIZED.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>The pause element being released must have been allocated with auth_level=IEA_AUTHORIZED.</td>
</tr>
</tbody>
</table>

target_du_pause_element_token
Supplied parameter
  Type: Character string
  Character Set: N/A
  Length: 16 bytes

Contains the pause element token that identifies the pause element used to
pause the task. If the PET identifies a pause element that has not been paused
(that is, the task has not been paused), the task will not be paused. However,
the value specified in target_du_release_code will be returned to the caller of
Pause.

target_du_release_code
Supplied parameter
  Type: Character string
  Character Set: N/A
  Length: 3 bytes

Contains the release code returned to the caller of Pause or Transfer service
that used (or will use) the same PET to pause a task. If your program is not
using this code for communication, set this field to zero.

ABEND codes
None.

Return codes
When the service returns control to the resource manager, GPR 15 and return_code
contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
</tr>
<tr>
<td>Equate symbol</td>
<td></td>
</tr>
<tr>
<td>00 (00)</td>
<td>Meaning: Successful completion.</td>
</tr>
<tr>
<td>IEA_SUCCESS</td>
<td>Action: None.</td>
</tr>
</tbody>
</table>
### IEAVRLS callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex) Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04 (04) IEA_PE_TOKEN_BAD                   | **Meaning:** The specified pause element token is not valid. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 08 (08) IEA_PE_TOKEN_STALE                 | **Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET be returned on Pause or Transfer.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 16 (10) IEA_SLEEP_DISRUPTED                | **Meaning:** RTM has terminated the task; no release is necessary.  
**Action:** None |
| 20 (14) IEA_SPACE_TERMINATING              | **Meaning:** The address space that contains the task that is terminating; no release is necessary.  
**Action:** None |
| 24 (18) IEA_LOCK_HELD                      | **Meaning:** Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 32 (20) IEA_PE_BAD_STATE                   | **Meaning:** Program error. The pause element associated with the pause element token specified is invalid or has already been prereleased.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 36 (24) IEA_UNSUPPORTED_MVS_RELEASE         | **Meaning:** Environmental error. The system release does not support this service. The system rejects the service call.  
**Action:** Run the program on a system that supports the service. |
| 40 (28) IEA_INVALID_AUTHCODE               | **Meaning:** Program error. The auth_level value specified in the call is not valid. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 44 (2C) IEA_INVALID_MODE                   | **Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 60 (3C) IEA_AUTH_TOKEN                     | **Meaning:** Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call.  
**Action:** Program error. The specified pause element token is not valid. The system rejects the service call. |
## IEAVRLS callable service

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
</tr>
<tr>
<td>Equate symbol</td>
<td></td>
</tr>
<tr>
<td>64 (40)</td>
<td><strong>IEA_PF_NOT_HOME</strong></td>
</tr>
</tbody>
</table>
|                | **Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address.  
|                | **Action**: Check the calling program for a probable coding error. Correct the program and rerun it. |
| 4095 (FFF)     | **IEA_UNEXPECTED_ERROR** |
|                | **Meaning**: This service routine encountered an unexpected error. The system rejects this service request.  
|                | **Action**: Contact IBM support. |
IEAVRLS callable service
Chapter 55. IEAVRLS2 — Release

Description

Call Release to remove a task or SRB that has been paused, or to keep a task or SRB from being paused.

Although a pause element can be used multiple times to pause a task or SRB, a pause element token can be used to successfully pause and release a task or SRB only once. Each time a pause element is used, the system generates a new PET to identify the pause element. The system returns the new updated PET on calls to the Pause and Transfer services.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=BRANCH, supervisor state and PSW key 0.</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC:</td>
</tr>
<tr>
<td></td>
<td>– Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7.</td>
</tr>
<tr>
<td></td>
<td>– Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=BRANCH: Task or SRB</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC:</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC: PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit addressing mode.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=BRANCH: The CPU, CMS, or local locks may be held.</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC: No locks may be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>
IEAVRLS2 callable service

Restrictions
Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Key 1-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC.

Release cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information
Before calling the Release service, the caller must ensure that the following general purpose (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 72-byte register save area.</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.
**IEAVRLS2 callable service**

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEAVRLS2</td>
<td>(return_code, target_du_pause_element_token, target_du_release_code, linkage)</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

- **return_code**
  - Returned parameter
  - Type: Integer
  - Character Set: N/A
  - Length: 4 bytes
  - Contains the return from the Release service.

- **target_du_pause_element_token**
  - Supplied parameter
  - Type: Character string
  - Character Set: N/A
  - Length: 16 bytes
  - Contains the pause element token that identifies the pause element used to pause a task or SRB. You obtain the PET from the Allocate_Pause_Element service.
  - When you use a PET in a call to the Pause service, you cannot reuse the PET on a second call to Pause or on a call to Transfer. The Pause service returns a new PET in updated_pause_element_token. The new PET now identifies the pause element used to pause the task or SRB; use the new PET the next time you make a Pause request using the same Pause element.

- **target_du_release_code**
  - Supplied parameter
  - Type: Character string
  - Character Set: N/A
  - Length: 3 bytes
  - Contains the release code returned to the caller of Pause or Transfer service that used (or will use) the same PET to pause a task or SRB. If your program is not using this code for communication, set this field to zero.

- **linkage**
  - Supplied parameter
  - Type: Integer
  - Character Set: N/A
  - Length: 4 bytes
  - Specifies how the Release service routine is to be invoked. The following options are supported:
IEAVRLS2 callable service

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_SVC</td>
<td>0</td>
<td>The Release service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and either problem state or supervisor state.</td>
</tr>
<tr>
<td>IEA_LINKAGE_BRANCH</td>
<td>1</td>
<td>The Release service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.</td>
</tr>
</tbody>
</table>

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return code: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>IEA_SUCCESS</td>
<td><strong>Meaning:</strong> Successful completion. <strong>Action:</strong> None.</td>
</tr>
<tr>
<td>04 (04)</td>
<td>IEA_PE_TOKEN_BAD</td>
<td><strong>Meaning:</strong> The specified pause element token is not valid. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>IEA_PE_TOKEN_STALE</td>
<td><strong>Meaning:</strong> The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>16 (10)</td>
<td>IEA_SLEEP_DISRUPTED</td>
<td><strong>Meaning:</strong> RTM has terminated the task or SRB; no release is necessary. <strong>Action:</strong> None</td>
</tr>
<tr>
<td>20 (14)</td>
<td>IEA_SPACE_TERMINATING</td>
<td><strong>Meaning:</strong> The address space that contains the task or SRB is terminating; no release is necessary. <strong>Action:</strong> None</td>
</tr>
<tr>
<td>24 (18)</td>
<td>IEA_LOCK_HELD</td>
<td><strong>Meaning:</strong> Program error. The caller is holding one or more locks; other than the local lock, CMS, or CPU lock, no locks may be held. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>32 (20)</td>
<td>IEA_PE_BAD_STATE</td>
<td><strong>Meaning:</strong> Program error. The pause element associated with the pause element token specified is invalid or has already been prereleased. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>IEA_UNSUPPORTED_MVS_RELEASE</td>
<td><strong>Meaning:</strong> Environmental error. The system release does not support this service. The system rejects the service call. <strong>Action:</strong> Run the program on a system that supports the service.</td>
</tr>
</tbody>
</table>
## IEAVRLS2 callable service

<table>
<thead>
<tr>
<th>Return code: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 44 (2C)                   | IEA_INVALID_MODE       | **Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 60 (3C)                   | IEA_AUTH_TOKEN         | **Meaning:** Program error. The caller was in Problem state or key 8, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call.  
**Action:** Program error. The specified pause element token is not valid. The system rejects the service call. |
| 64 (40)                   | IEA_PE_NOT_HOME        | **Meaning:** Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 84 (54)                   | IEA_INVALID_LINKAGE    | **Meaning:** Program error. The linkage value specified is not valid. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 4095 (FFF)                | IEA_UNEXPECTED_ERROR   | **Meaning:** This service routine encountered an unexpected error. The system rejects this service request.  
**Action:** Contact IBM support. |
IEAVRLS2 callable service
Description

Call Retrieve_Pause_Element_Information to get information about a pause element. The information returned includes:

- Its authorization level
- The address space that currently owns it
- Its current state (Reset, Prereleased, Paused, or Released)
- If its state is Prereleased or Released, its Release Code

An authorized program can use Retrieve_Pause_Element_Information to test the validity of a pause element passed by an unauthorized program. The authorized program can do this to ensure that it does not perform any operation, such as releasing the pause element, unless the unauthorized program is also able to perform the same operation.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state and any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>• When supervisor state and PSW key 0: Task or SRB</td>
</tr>
<tr>
<td></td>
<td>• When problem state, or not PSW key 0: Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>• When supervisor state and PSW key 0: Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td></td>
<td>• When problem state, or not PSW key 0:</td>
</tr>
<tr>
<td></td>
<td>• PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit addressing mode.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

None.
IEAVRPI callable service

Input register information
Before calling the Retrieve_Pause_Element_Information service, the caller does not need to place any information into any register, unless using it in register notation for the parameters, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEAVRPI</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained as follows:

**return_code**
- Returned parameter
- Type: Integer
- Character Set: N/A
IEAVRPI callable service

- Length: 4 bytes
  Contains the return code from the Retrieve_Pause_Element_Information service.

**auth_level**
Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the caller's authorization level. The following levels are supported: IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which can be used by the calling program.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>The caller is not key 0 and supervisor state.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>The caller is both key 0 and supervisor state.</td>
</tr>
</tbody>
</table>

**pause_element_token**
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element for which information will be returned. You obtain the PET from the Allocate_Pause_Element service.

**authorization**
Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The authorization level of the creator of the pause element specified by the input PET.

One of the following values:

<table>
<thead>
<tr>
<th>IEAASM and IEAC defined constants</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>The caller is not key 0 and supervisor state.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>The caller is not key 0 and supervisor state.</td>
</tr>
<tr>
<td>IEA_UNAUTHORIZED + IEA_CHECKPOINTOK</td>
<td>2</td>
<td>Unauthorized PET that can tolerate the pause elements' not being restored upon a restart after a checkpoint.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED + IEA_CHECKPOINTOK</td>
<td>3</td>
<td>Authorized PET that can tolerate the pause elements' not being restored upon a restart after a checkpoint.</td>
</tr>
</tbody>
</table>

**owner**
Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 8 bytes
IEAVRPI callable service

The Stoken of the address space that currently owns the pause element specified by the input PET.

\[ \text{state} \]

Return parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The state of the pause element specified by the input PET.

**Note:** The value returned is the state at the time the service obtained it. The state may have changed after it was obtained.

<table>
<thead>
<tr>
<th>State Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAV_PET_PRERELEASE</td>
<td>The PE was released before any task or SRB was suspended on it, and no task or SRB has attempted to pause it.</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>IEAV_PET_RESET</td>
<td>The PE is not being used to make any task or SRB nondispatchable. If the PE is used in an attempt to pause the current task or SRB, the task or SRB will be made nondispatchable.</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>IEAV_PET_RELEASED</td>
<td>The task RB or SRB is currently dispatchable, but control has not been returned to the task or SRB following a call to the Pause or Transfer service. A call to the Release or Transfer service has released the task or SRB. In either case, control has not been returned to the caller of the Pause or Transfer service. The system has not transitioned the PE into the RESET state.</td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>IEAV_PET_PAUSED</td>
<td>A task RB or SRB is currently nondispatchable. Its dispatchability is controlled by the PE.</td>
</tr>
<tr>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{release_code} \]

Return parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code, specified by the issuer of the Release service. A Release that specified this code released the task or SRB from its paused condition.

**Note:** The returned value is random if the state parameter is not IEAV_PET_RELEASED or IEAV_PET_PRERELEASED.

**ABEND codes**

None.

**Return codes**

When the service returns control to your program, GPR 15 contains one of the following return codes:
<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>IEA_SUCCESS</td>
<td><strong>Meaning:</strong> Successful completion.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None</td>
</tr>
<tr>
<td>04 (04)</td>
<td>IEA_PE_TOKEN_BAD</td>
<td><strong>Meaning:</strong> Program error. The specified pause element token is not valid. The</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>program and rerun it.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>IEA_PE_TOKEN_STALE</td>
<td><strong>Meaning:</strong> The specified pause element token is stale; that is, it was valid but</td>
</tr>
<tr>
<td></td>
<td></td>
<td>has been used on the Pause or Transfer service. This service requires the updated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PET returned on Pause or Transfer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>program and rerun it.</td>
</tr>
<tr>
<td>24 (18)</td>
<td>IEA_LOCK_HELD</td>
<td><strong>Meaning:</strong> Program error. The caller is holding one or more locks; no locks must</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be held. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>program and rerun it.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>IEA_UNSUPPORTED_MVS_RELEASE</td>
<td><strong>Meaning:</strong> Environmental error. The system release does not support this service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Run the program on a system that supports the service.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>IEA_INVALID_MODE</td>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this service requires. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>program and rerun it.</td>
</tr>
<tr>
<td>60 (3C)</td>
<td>IEA_AUTH_TOKEN</td>
<td><strong>Meaning:</strong> Program error. The caller specified an unauthorized auth_level type,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>but a pause element token allocated with an authorized auth_level type was</td>
</tr>
<tr>
<td></td>
<td></td>
<td>encountered. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Program error. The specified pause element token is not valid. The</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system rejects the service call.</td>
</tr>
<tr>
<td>64 (40)</td>
<td>IEA_PE_NOT_HOME</td>
<td><strong>Meaning:</strong> Program error. The caller specified an unauthorized auth_level type,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>but a pause element token for a pause element allocated to another address space</td>
</tr>
<tr>
<td></td>
<td></td>
<td>was specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>program and rerun it.</td>
</tr>
</tbody>
</table>
IEAVRPI callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 4095 (FFF)                    | IEA_UNEXPECTED_ERROR       | **Meaning**: This service routine encountered an unexpected error. The system rejects this service request.  
**Action**: Contact IBM support. |
Chapter 57. IEAVRPI2 — Retrieve_Pause_Element_Information

service

Description

Call Retrieve_Pause_Element_Information to get information about a pause element. The information returned includes:

- Its authorization level
- Its current state (Reset, Prereleased, Paused, or Released)
- If its state is Prereleased or Released, its Release Code
- The stoken of the owner of the pause element (See IEAVAPE — Allocate_Pause_Element for details of ownership).
- The stoken of the home address space of the task or SRB which is paused by the pause element.

An authorized program can use Retrieve_Pause_Element_Information to test the validity of a pause element passed by an unauthorized program. The authorized program may do this to ensure that it does not perform any operation, such as releasing the pause element, unless the unauthorized program is also able to perform the same operation.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>• For LINKAGE=BRANCH, supervisor state and PSW key 0.</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC:</td>
</tr>
<tr>
<td></td>
<td>• Working with an IEA_AUTHORIZED pause element,</td>
</tr>
<tr>
<td></td>
<td>supervisor state and PSW key 0-7.</td>
</tr>
<tr>
<td></td>
<td>• Working with an IEA_UNAUTHORIZED pause element, problem state and any</td>
</tr>
<tr>
<td></td>
<td>PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>• For LINKAGE=BRANCH: Task or SRB</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC: Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC: PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit addressing mode.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the calling program’s object code with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:
IEAVRPI2 callable service

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Key 2-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC.

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Input register information

Before calling the Retrieve_Pause_Element_Information service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 72-byte register save area</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.
Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEAVRPI2</td>
<td>returns the return code from the Retrieve_Pause_Element_Information service.</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

**return_code**
- Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Retrieve_Pause_Element_Information service.

**pause_element_auth_level**
- Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the authorization level with which the pause element specified by the input PET was allocated. The following levels are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_PET_UNAUTHORIZED</td>
<td>0</td>
<td>The pause element was allocated with pause_element_auth_level=IEA_PET_UNAUTHORIZED.</td>
</tr>
<tr>
<td>IEA_PET_AUTHORIZED</td>
<td>1</td>
<td>The pause element was allocated with pause_element_auth_level=IEA_PET_AUTHORIZED.</td>
</tr>
</tbody>
</table>

**pause_element_token**
- Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element for which information will be returned. You obtain the PET from the Allocate_Pause_Element service.

**linkage**
- Supplied parameter
- Type: Integer
- Character Set: N/A
The calling program can use the constants defined in IEAASM or IEAC, as appropriate. Add the specified values together to achieve the desired results. For example, to specify linkage branch and untrusted PET, specify `IEA_LINKAGE_BRANCH + IEA_UNTRUSTED_PET`.

The following options are supported:

**Table 60. Linkage variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_SVC</td>
<td>0</td>
<td>The <code>Retrieve_Pause_Element_Information</code> service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and in either problem state or supervisor state.</td>
</tr>
<tr>
<td>IEA_LINKAGE_BRANCH</td>
<td>1</td>
<td>The <code>Retrieve_Pause_Element_Information</code> service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.</td>
</tr>
</tbody>
</table>

**Table 61. Untrusted attribute linkage variable**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNTRUSTED_PET</td>
<td>2</td>
<td>The <code>Retrieve_Pause_Element_Information</code> service routine is to validate that the input PET is allowed to be used by an unauthorized program. An authorized program should use this if it is validating a PET provided to it by an unauthorized caller.</td>
</tr>
</tbody>
</table>

{returned parameter}

Type: Character string

Character Set: N/A

Length: 8 bytes

Specifies the stoken of the address space that currently owns the pause element specified by the input PET. The owner of a PE allocated by IEAVAPE2 is static and specified on the IAVEAPE2 call. The owner of a PE allocated by IEAVAPE is dynamic. See IEAVAPE — Allocate_Pause_Element for details.

{current_stoken}

Returned parameter

Type: Character string

Character Set: N/A

Length: 8 bytes

If the value returned in state is IEA_PET_PAUSED, The stoken of the home address space of the task or SRB which is paused by the specified pause element. If the value in state is not IEA_PET_PAUSED, the information returned in this parameter is undefined.
IEAVRPI2 callable service

state
Return parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The state of the pause element specified by the input PET.

Note: The value returned is the state at the time the service obtained it. The state may have changed after it was obtained.

<table>
<thead>
<tr>
<th>State Constant</th>
<th>Hexadecimal (Decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAV_PET_PRERELEASE</td>
<td>1 (1)</td>
<td>The PE was released before any task or SRB was suspended on it, and no task or SRB has attempted to pause it.</td>
</tr>
<tr>
<td>IEAV_PET_RESET</td>
<td>2 (2)</td>
<td>The PE is not being used to make any task or SRB nondispatchable. If the PE is used in an attempt to pause the current task or SRB, the task or SRB will be made nondispatchable.</td>
</tr>
<tr>
<td>IEAV_PET_RELEASED</td>
<td>40 (64)</td>
<td>The task RB or SRB is currently dispatchable, but control has not been returned to the task or SRB following a call to the Pause or Transfer service. A call to the Release or Transfer service has released the task or SRB. In either case, control has not been returned to the caller of the Pause or Transfer service. The system has not transitioned the PE into the RESET state.</td>
</tr>
<tr>
<td>IEAV_PET_PAUSED</td>
<td>80 (128)</td>
<td>A task RB or SRB is currently nondispatchable. Its dispatchability is controlled by the PE.</td>
</tr>
</tbody>
</table>

release_code
Return parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code, specified by the issuer of the Release service. A Release that specified this code released the task or SRB from its paused condition.

Note: The returned value is random if the state parameter is not IEAV_PET_RELEASED or IEAV_PET_PRERELEASE.

ABEND codes
None.

Return codes
When the service returns control to your program, GPR 15 contains one of the following return codes:
### IEAVRPI2 callable service

<table>
<thead>
<tr>
<th>Return code: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>IEA_SUCCESS</td>
<td>Meaning: Successful completion.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None</td>
</tr>
<tr>
<td>04 (04)</td>
<td>IEA_PE_TOKEN_BAD</td>
<td><strong>Meaning:</strong> Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>IEA_PE_TOKEN_STALE</td>
<td><strong>Meaning:</strong> The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>24 (18)</td>
<td>IEA_LOCK_HELD</td>
<td><strong>Meaning:</strong> Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>IEA_UNSUPPORTED_MVS_RELEASE</td>
<td><strong>Meaning:</strong> Environmental error. The system release does not support this service. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Run the program on a system that supports the service.</td>
</tr>
<tr>
<td>40 (28)</td>
<td>IEA_INVALID_AUTHCODE</td>
<td><strong>Meaning:</strong> Program error. The pause_element_auth_level value specified in the call is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>IEA_INVALID_MODE</td>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>60 (3C)</td>
<td>IEA_AUTH_LEVEL_MISMATCH</td>
<td><strong>Meaning:</strong> Program error. The caller was in Problem state or key 8, or specified IEA_UNTRUSTED_PET, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td>84 (54)</td>
<td>IEA_INVALID_LINKAGE</td>
<td><strong>Meaning:</strong> Program error. The linkage value specified is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>4095 (FFF)</td>
<td>IEA_UNEXPECTED_ERROR</td>
<td><strong>Meaning:</strong> This service routine encountered an unexpected error. The system rejects this service request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Contact IBM support.</td>
</tr>
</tbody>
</table>
Chapter 58. IEAVTPE — Test_Pause_Element service

Description

Call Test_Pause_Element to test a pause element and determine its state. If its state is Prereleased or Released, the pause element's release code will also be returned.

To ensure minimal overhead when you use the service, Test_Pause_Element establishes no recovery. You are responsible for supplying any needed recovery to handle errors that occur due to invalid input pause element Tokens or call state errors.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>• When supervisor state and PSW key 0: Task or SRB</td>
</tr>
<tr>
<td></td>
<td>• When problem state, or not PSW key 0: Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit addressing mode.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

None.

Input register information

Before calling the Test_Pause_Element service, the caller does not have to place any information into any register, unless using it in register notation for the parameters, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IEAVTPE callable service

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>14</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEAVTPE</td>
<td>(return_code</td>
</tr>
<tr>
<td></td>
<td>,pause_element_token</td>
</tr>
<tr>
<td></td>
<td>,state</td>
</tr>
<tr>
<td></td>
<td>,release_code)</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

**return_code**

Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes
  Contains the return code from the Test_Pause_Element service.

**pause_element_token**

Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes
  A pause element token that identifies the pause element for which information is to be returned. You obtain the PET from the Allocate_Pause_Element service.
IEAVTPE callable service

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The state of the pause element specified by the input PET.

Note: The value returned is the state at the time the service obtained it. The state may have changed after it was obtained.

<table>
<thead>
<tr>
<th>State Constant Hexadecimal (Decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAV_PET_PRERELEASE 1 (1)</td>
<td>The PE was released before any task or SRB was suspended on it, and no task or SRB has attempted to pause it.</td>
</tr>
<tr>
<td>IEAV_PET_RESET 2 (2)</td>
<td>The PE is not being used to make any task or SRB nondispatchable. If the PE is used in an attempt to pause the current task or SRB, the task or SRB will be made nondispatchable.</td>
</tr>
<tr>
<td>IEAV_PET_RELEASED 40 (64)</td>
<td>The task RB or SRB is currently dispatchable, but control has not been returned to the task or SRB following a call to the Pause or Transfer service. A call to the Release or Transfer service has released the task or SRB. In either case, control has not been returned to the caller of the Pause or Transfer service. The system has not transitioned the PE into the RESET state.</td>
</tr>
<tr>
<td>IEAV_PET_PAUSED 80 (128)</td>
<td>A task RB or SRB is currently nondispatchable. Its dispatchability is controlled by the PE.</td>
</tr>
</tbody>
</table>

```
release_code
```

Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code, specified by the issuer of the Release service. A Release that specified this code released the task or SRB from its paused condition.

Note: The returned value is random if the state parameter is not IEAV_PET_RELEASED or IEAV_PET_PRERELEASED.

**ABEND codes**

None.

**Return codes**

When the service returns control to your program, GPR 15 contains one of the following return codes:
### IEAVTPE callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>IEA_SUCCESS</td>
<td><strong>Meaning:</strong> Successful completion. <strong>Action:</strong> None</td>
</tr>
<tr>
<td>04 (04)</td>
<td>IEA_PE_TOKEN_BAD</td>
<td><strong>Meaning:</strong> Program error. The specified pause element token is not valid. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>IEA_PE_TOKEN_STALE</td>
<td><strong>Meaning:</strong> The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
</tbody>
</table>
Chapter 59. IEAVXFR — Transfer service

Description
Call the Transfer service to release a paused task, and, when possible, give it immediate control. This service can also, optionally, pause the task under which the Transfer request is made. If the caller does not request that its task be paused, the caller's task remains dispatchable.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>• When supervisor state and PSW key 0: Task or SRB.</td>
</tr>
<tr>
<td></td>
<td>• When problem state, or not PSW key 0: Task.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN</td>
</tr>
<tr>
<td></td>
<td>• For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit addressing mode.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>When supervisor state and PSW key 0 and a current_du_pause_element_token of 16 bytes of binary zeros are specified, the local lock may be held. Otherwise, no locks may be held.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements
Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions
When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only transfer to another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information
Before calling the Transfer service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:
IEAVXFR callable service

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
</table>
1 | Address of the parameter address list. |
13 | Address of a 72-byte register save area. |

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
</table>
0-1 | Used as work registers by the system |
2-13 | Unchanged |
14 | Used as a work register by the system |
15 | Return code |

When control returns to the caller, the access registers (ARs) contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
</table>
0-1 | Used as work registers by the system |
2-14 | Unchanged |
15 | Used as a work register by the system |

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEAVXFR</td>
<td>,return_code,auth_level,current_du_pause_element_token,updated_pause_element_token,current_du_release_code,target_du_pause_element_token,target_du_release_code</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

return_code

- Returned parameter
- Type: Integer
IEAVXFR callable service

- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Transfer service.

,.auth_level
Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum authorization level of the pause element being deallocated. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEAAUTHORIZED, which the calling program can use. The following levels are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (HEX)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>The pause elements must have been allocated with auth_level=UNAUTHORIZED.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>The pause elements must have been allocated with auth_level=AUTHORIZED.</td>
</tr>
</tbody>
</table>

,.current_du_pause_element_token
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies the pause element used to pause the current task. Once a PET is used on a call to the Pause service, it cannot be reused on a second call to Pause or as a current_du_pause_element_token on Transfer. A new PET is returned to updated_pause_element_token. The new PET now properly defines the pause element and should be used the next time a pause, transfer, release, or deallocate_pause_element request is made using the same pause element.

If the value specified is 16-bytes of binary zeros, the current task will not be paused. The updated_pause_element_token and current_du_release_code will be unpredictable.

CAUTION:
Do not specify the same PET for both current_du_pause_element_token and target_pause_element_token.

,.updated_pause_element_token
Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a new pause element token that identifies the pause element originally identified by the PET specified in current_du_pause_element_token. The PET originally specified in current_du_pause_element_token cannot be reused after a successful call to Pause or Transfer.

If you set the current_du_pause_element_token to zeros, the contents of updated_pause_element_token are unpredictable.
IEAVXFR callable service

,,current_du_release_code
  Returned parameter
  • Type: Character string
  • Character Set: N/A
  • Length: 3 bytes
  
  Contains the release code set by the issuer of the Release or Transfer service that released the current task from its paused condition.

  If you set the current_du_pause_element_token to zero, the contents are unpredictable.

,,target_du_pause_element_token
  Supplied parameter
  • Type: Character string
  • Character Set: N/A
  • Length: 16 bytes
  
  Contains a pause element token that identifies the pause element to release the target task. Any PET that specifies a pause element not currently being used to pause a task is valid. When a PET for a previously released pause element is used to try to pause a task, the task is not paused; however, the value specified in target_du_release_code will still be returned to the caller of Pause or Transfer.

  If the task was paused and is now dispatchable, the task will immediately be given control on the current processor.

  CAUTION:
  Do not use the same PET for both current_du_pause_element_token and target_du_pause_element_token.

,,target_du_release_code
  Supplied parameter
  • Type: Character string
  • Character Set: N/A
  • Length: 3 bytes
  
  Contains the release code returned to the issuer of the Pause or Transfer service that is used (or will use) the same PET to pause a task.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return Code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>IEA_SUCCESS</td>
<td>Meaning: Successful completion. Action: None</td>
</tr>
</tbody>
</table>
IEAVXFR callable service

<table>
<thead>
<tr>
<th>Return Code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04 (04)                      | IEA_PE_TOKEN_BAD             | **Meaning:** Program error. The specified pause element token is not valid. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 08 (08)                      | IEA_PE_TOKEN_STALE           | **Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 12 (0C)                      | IEA_DUPLICATE_PAUSE          | **Meaning:** The work unit has already been paused using the specified pause element token. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 16 (10)                      | IEA_SLEEP_DISRUPTED          | **Meaning:** RTM has terminated the task or SRB; no release is necessary.  
**Action:** None |
| 20 (14)                      | IEA_SPACE_TERMINATING        | **Meaning:** The address space that contains the task or SRB is terminating; no release is necessary.  
**Action:** None |
| 24 (18)                      | IEA_LOCK_HELD                | **Meaning:** Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 32 (20)                      | IEA_PE_BAD_STATE             | **Meaning:** Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it. |
| 36 (24)                      | IEA_UNSUPPORTED_MVS_RELEASE   | **Meaning:** Environmental error. The system release does not support this service. The system rejects the service call.  
**Action:** Run the program on a system that supports the service. |
### IEAVXFR callable service

<table>
<thead>
<tr>
<th>Return Code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 (28)</td>
<td>IEA_INVALID_AUTHCODE</td>
<td><strong>Meaning:</strong> Program error. The auth_level value specified in the call is not valid. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>IEA_INVALID_MODE</td>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>52 (34)</td>
<td>IEA_ALREADY_SUSPENDED</td>
<td><strong>Meaning:</strong> The pause element was already paused. <strong>Action:</strong> Check the calling program for a probable coding error and correct the program and rerun it.</td>
</tr>
<tr>
<td>60 (3C)</td>
<td>IEA_AUTH_TOKEN</td>
<td><strong>Meaning:</strong> Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. <strong>Action:</strong> Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td>64 (40)</td>
<td>IEA_PE_NOT_HOME</td>
<td><strong>Meaning:</strong> Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>68 (44)</td>
<td>IEA_XFER_TO_SELF</td>
<td><strong>Meaning:</strong> Program error. The specified current_du_pause_element_token and target_du_pause_element_token are the same. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>72 (48)</td>
<td>IEA_XFER_FAILED</td>
<td><strong>Meaning:</strong> The transfer failed, and the current_du_pause_element_token is no longer useable. <strong>Action:</strong> Reissue the transfer request using the updated_du_pause_element_token. Deallocate the current_du_pause_element_token.</td>
</tr>
<tr>
<td>4095 (FFF)</td>
<td>IEA_UNEXPECTED_ERROR</td>
<td><strong>Meaning:</strong> This service routine encountered an unexpected error. The system rejects this service request. <strong>Action:</strong> Contact IBM support.</td>
</tr>
</tbody>
</table>
Chapter 60. IEAVXFR2 — Transfer service

Description

Call the Transfer service to release a paused task or SRB, and, when possible, give it immediate control. This service can also, optionally, pause the task or SRB under which the Transfer request is made. If the caller does not request that its task or SRB be paused, the caller’s task or SRB remains dispatchable.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>• For LINKAGE=BRANCH, supervisor state and PSW key 0.</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC:</td>
</tr>
<tr>
<td></td>
<td>– Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7.</td>
</tr>
<tr>
<td></td>
<td>– Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>• For LINKAGE=BRANCH: Task or SRB</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC: Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC: PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit addressing mode.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>If LINKAGE=BRANCH and a current_du_pause_element_token of 16 bytes of binary zeros is specified, the local lock may be held. Otherwise, no locks may be held.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.
IEAVXFR2 callable service

Key 1-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC.

Transfer cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

**Input register information**

Before calling the Transfer service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 72-byte register save area.</td>
</tr>
</tbody>
</table>

**Output register information**

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.
CALL IEAVXFR2

### Parameters

The parameters are explained as follows:

**return_code**
- Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes
- Contains the return code from the Transfer service.

**current_du_pause_element_token**
- Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes
- Contains a pause element token that identifies the pause element that is being or will be used to pause a task or SRB. When a PET is used on a call to the Pause service, it cannot be reused on a second call to Pause or as a current_du_pause_element_token on Transfer. A new PET is returned to update_pause_element_token. The new PET properly defines the pause element and should be used the next time a pause, transfer, release, or deallocate_pause_element request is made using the same pause element.
- If the value specified is 16-bytes of binary zeros, the current task or SRB is not paused. The updated_pause_element_token and current_du_release_code will be unpredictable.

**CAUTION:**
- Do not specify the same PET for both current_du_pause_element_token and target_du_pause_element_token.

**updated_pause_element_token**
- Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes
- Contains a new pause element token that identifies the pause element originally identified by the PET specified in current_du_pause_element_token. The PET originally specified in current_du_pause_element_token cannot be reused after a successful call to Pause or Transfer.
IEAVXFR2 callable service

If you set the current_du_pause_element_token to zeros, the contents of updated_pause_element_token are unpredictable.

$current_du_release_code$

Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code set by the issuer of the Release or Transfer service that released the current task or SRB from its paused condition.

If you set the current_du_pause_element_token to zero, the contents are unpredictable.

$target_du_pause_element_token$

Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies a pause element that is being or will be used to pause a task or SRB. If the task or SRB is paused, it will be released, and, if possible, be given control. If the task or SRB is not paused using the specified pause element, it will not be paused when an attempt to pause is made. In either case the task or SRB will be returned the value specified in target_du_release_code.

**CAUTION:**
Do not use the same PET for both current_du_pause_element_token and target_du_pause_element_token.

$target_du_release_code$

Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the issuer of the Pause or Transfer service used (or will use) the PET specified in target_du_pause_element_token to pause a task or SRB.

$linkage$

Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Transfer service routine is to be invoked. The following options are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_SVC</td>
<td>0</td>
<td>The Transfer service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and either problem state or supervisor state.</td>
</tr>
</tbody>
</table>
### IEAVXFR2 callable service

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_BRANCH</td>
<td>1</td>
<td>The Transfer service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.</td>
</tr>
</tbody>
</table>

### ABEND codes

None.

### Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return Code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00 (00)                       | IEA_SUCCESS           | **Meaning:** Successful completion.  
**Action:** None |
| 04 (04)                       | IEA_PE_TOKEN_BAD      | **Meaning:** Program error. The specified pause element token is not valid. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 08 (08)                       | IEA_PE_TOKEN_STALE    | **Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 12 (0C)                       | IEA_DUPLICATE_PAUSE   | **Meaning:** The work unit has already been paused using the specified pause element token. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 16 (10)                       | IEA_SLEEP_DISRUPTED   | **Meaning:** RTM has terminated the task or SRB; no release is necessary.  
**Action:** None |
| 20 (14)                       | IEA_SPACE_TERMINATING | **Meaning:** The address space that contains the task or SRB is terminating; no release is necessary.  
**Action:** None |
| 24 (18)                       | IEA_LOCK_HELD         | **Meaning:** Program error. If a current_du_pause_element_token of 16 bytes of binary zeros is specified, one or more locks other than the local lock are held. Otherwise, one or more locks are held. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
## IEAVXFR2 callable service

<table>
<thead>
<tr>
<th>Return Code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 (20)</td>
<td>IEA_PE_BAD_STATE</td>
<td><strong>Meaning:</strong> Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>IEA_UNSUPPORTED_MVS_RELEASE</td>
<td><strong>Meaning:</strong> Environmental error. The system release does not support this service. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Run the program on a system that supports the service.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>IEA_INVALID_MODE</td>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>52 (34)</td>
<td>IEA_ALREADY_SUSPENDED</td>
<td><strong>Meaning:</strong> The pause element was already paused.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error and correct the program and rerun it.</td>
</tr>
<tr>
<td>60 (3C)</td>
<td>IEA_AUTH_TOKEN</td>
<td><strong>Meaning:</strong> Program error. The caller was in Problem state or key 8, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td>64 (40)</td>
<td>IEA_PE_NOT_HOME</td>
<td><strong>Meaning:</strong> Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>68 (44)</td>
<td>IEA_XFER_TO_SELF</td>
<td><strong>Meaning:</strong> Program error. The specified current_du_pause_element_token and target_du_pause_element_token are the same.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>72 (48)</td>
<td>IEA_XFER_FAILED</td>
<td><strong>Meaning:</strong> The transfer failed, and the current_du_pause_element_token is no longer usable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Reissue the transfer request using the updated_du_pause_element_token. Deallocate the current_du_pause_element_token.</td>
</tr>
<tr>
<td>84 (54)</td>
<td>IEA_INVALID_LINKAGE</td>
<td><strong>Meaning:</strong> Program error. The linkage value specified is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>4095 (FFF)</td>
<td>IEA_UNEXPECTED_ERROR</td>
<td><strong>Meaning:</strong> This service routine encountered an unexpected error. The system rejects this service request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Contact IBM support.</td>
</tr>
</tbody>
</table>
Chapter 61. IEA4APE — Allocate_Pause_Element

Description

Allocate_Pause_Element obtains a pause element token (PET), which uniquely identifies a pause element. The PET is used as input to the following services:

- Pause
- Release
- Transfer
- Deallocate_Pause_Element

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN</td>
</tr>
<tr>
<td></td>
<td>• For auth_level=IEAAUTHORIZED: Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>• When supervisor state and PSW key 0: The local lock may be held.</td>
</tr>
<tr>
<td></td>
<td>• When problem state, or not PSW key 0: No locks may be held.</td>
</tr>
</tbody>
</table>

Control parameters: Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.
**IEA4APE callable service**

**Input register information**
Before calling Allocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 144-byte register save area.</td>
</tr>
</tbody>
</table>

**Output register information**
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return Code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**
None.

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTATE AMODE64=YES</td>
<td></td>
</tr>
<tr>
<td>CALL IEA4APE</td>
<td>/return_code, /auth_level, /pause_element_token</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

- **return_code**
  Returned parameter
IEA4APE callable service

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Allocate_Pause_Element service.

\texttt{auth\_level}

Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Represents one or more possible levels of the pause element being allocated. The calling program can use the constants that are defined in IEAASM or IEAC. The level needed is derived by adding the values of the required types together. The authorization type is required.

For example, the level to allocate authorized pause elements that are checkpoint- or restart-tolerant is \texttt{IEA\_AUTHORIZED + IEA\_CHECKPOINTOK}, or 3.

The following levels are supported:

\begin{table}[h]
\centering
\begin{tabular}{|c|c|l|}
\hline
IEAASM and IEAC defined constants & Value (hexadecimal) & Meaning \tabularnewline \hline
\texttt{IEA\_UNAUTHORIZED} & 0 & When using the allocated pause element through other services, either \texttt{auth\_level = IEA\_UNAUTHORIZED} or \texttt{IEA\_AUTHORIZED} can be used. \tabularnewline \hline
\texttt{IEA\_AUTHORIZED} & 1 & When using the allocated pause element through other services, \texttt{auth\_level = IEA\_AUTHORIZED} will be required. Caller must be both key 0 and supervisor state. \tabularnewline \hline
\end{tabular}
\caption{Authorization}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|l|}
\hline
IEAASM and IEAC defined constants & Value (hexadecimal) & Meaning \tabularnewline \hline
\texttt{IEA\_CHECKPOINTOK} & 2 & The application can tolerate the pause elements not being restored upon a restart after a checkpoint. \tabularnewline \hline
\end{tabular}
\caption{Checkpoint/Restart Toleration - only available when the CVTPAUS4 bit is set in the CVT.}
\end{table}

\textbf{Note:} If the \texttt{IEA\_CHECKPOINTOK} value is not added to the authorization value, checkpoints cannot be taken when an allocated pause element exists.

\texttt{pause\_element\_token}

Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element that you can use to synchronize the processing of a task.
**IEA4APE callable service**

**ABEND codes**

None.

**Return codes**

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
</tr>
<tr>
<td>Equate symbol</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td></td>
</tr>
<tr>
<td>Meaning</td>
<td></td>
</tr>
</tbody>
</table>

| 00 (0) IEA_SUCCESS | Meaning: Successful completion. |
| Action: None.     | |

| 24 (18) IEA_LOCK_HELD | Meaning: Program error. If the auth_level indicates AUTHORIZED, locks other than the local lock are held. If the auth_level indicates UNAUTHORIZED, locks are held. The system rejects the service call. |
| Action: Check the calling program for a probable coding error. Correct the program and rerun it. |

| 36 (24) IEA_UNSUPPORTED_MVS_RELEASE | Meaning: Environmental error. The system release does not support this service. The system rejects the service call. |
| Action: Run the program on a system that supports the service. |

| 40 (28) IEA_PE_NOT_HOME | Meaning: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. |
| Action: Check the calling program for a probable coding error. Correct the program and rerun it. |

| 44 (2C) IEA_XFER_TO_SELF | Meaning: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. |
| Action: Check the calling program for a probable coding error. Correct the program and rerun it. |

| 48 (30) IEA_XFER_FAILED | Meaning: Environmental error. The system could not obtain storage for a pause element. The system rejects the service call. |
| Action: Retry the request later. If the problem persists, consult your system programmer. |

| 56 (38) IEA_NO_PETS_AVAILABLE | Meaning: There are no pause element tokens available. |
| Action: Retry the request later. |

| 4095 (FFF) IEA_UNEXPECTED_ERROR | Meaning: This service routine encountered an unexpected error. The system rejects this service request. |
| Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
Chapter 62. IEA4APE2 — Allocate_Pause_Element

Description

Allocate_Pause_Element obtains a pause element token (PET), which uniquely identifies a pause element. The PET is used as input to the following services:

- Pause
- Release
- Transfer
- Deallocate_Pause_Element

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>• For LINKAGE=BRANCH, supervisor state and PSW key 0.</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC:</td>
</tr>
<tr>
<td></td>
<td>-- Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7.</td>
</tr>
<tr>
<td></td>
<td>-- Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>• For LINKAGE=BRANCH: Task or SRB</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC:</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC: PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>• For LINKAGE=BRANCH: The local lock may be held.</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC: No locks may be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB) or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>
IEA4APE2 callable service

Restrictions
Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Allocate_Pause_Element cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information
Before calling Allocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 144-byte register save area.</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTATE AMODE64=YES</td>
<td></td>
</tr>
</tbody>
</table>
IEA4APE2 callable service

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEA4APE2</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

**return_code**

- Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Allocate_Pause_Element service.

**pause_element_auth_level**

- Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Represents one or more possible levels of the pause element being allocated. The calling program can use the constants defined in IEAASM or IEAC, as appropriate. The level desired results from adding the values of the required types together. The authorization type is not optional.

For instance, the level to allocate authorized pause elements that are checkpoint/restart tolerant is IEA_AUTHORIZED + IEA_CHECKPOINTOK, or 3.

The following levels are supported:

<table>
<thead>
<tr>
<th>IEAASM and IEAC defined constants</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>When using the allocated pause element through other services, either <code>pause_element_auth_level</code> IEA_UNAUTHORIZED or IEA_AUTHORIZED can be used.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>When using the allocated pause element through other services, <code>pause_element_auth_level</code> =IEA_AUTHORIZED is required. Caller must be both key 0 and supervisor state.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IEAASM and IEAC defined constants</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_CHECKPOINTOK</td>
<td>2</td>
<td>The application can tolerate the pause elements’ not being restored upon a restart after a checkpoint.</td>
</tr>
</tbody>
</table>

**Note:** If the IEA_CHECKPOINTOK value is not added to the authorization value, checkpoints cannot be taken when an allocated pause element exists.
IEA4APE2 callable service

`pause_element_token`
Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies a pause element that you can use to synchronize the processing of a task or SRB.

`pause_element_owner_stoken`
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 8 bytes

Specifies the space token (STOKEN) of the address space which is to be considered the owner of the Pause Element being allocated. Specify one of the following values:
- Binary zero: indicate the system should make the current primary address space the owner of the Pause Element. This is the only value valid for key 8-15 problem state callers.
- A valid STOKEN, indicate the system should make the address space with the matching STOKEN the owner for the pause element.

When the CMRO task (the first job step task) of an address space terminates, the system will release and deallocate any pause elements owned by the CMRO task's home address space. The table below describes exactly when the system will release and/or deallocate a Pause Element:

<table>
<thead>
<tr>
<th>Allocation Service version:</th>
<th>Deallocation Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA4APE</td>
<td>The PE will be deallocated by the system when one of the following events occurs:</td>
</tr>
<tr>
<td></td>
<td>• The PE was never used to pause a task or SRB and the CMRO task for the space which allocated it terminates.</td>
</tr>
<tr>
<td></td>
<td>• The PE is being used to pause a task or SRB which is asynchronously terminated via CALLRTM TYPE=ABTERM (for example, cancel or detach) or a PURGEDQ.</td>
</tr>
<tr>
<td></td>
<td>• The CMRO task of the home address space of the task or SRB which last used the PE terminates and the PE is not being used to pause an SRB.</td>
</tr>
<tr>
<td></td>
<td>The home address space of the task or SRB which last used the PE terminates</td>
</tr>
</tbody>
</table>
IEA4APE2 callable service

<table>
<thead>
<tr>
<th>Allocation Service version</th>
<th>Deallocation Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA4APE2</td>
<td>The PE will be deallocated by the system when one of the following events occurs:</td>
</tr>
<tr>
<td></td>
<td>• The CMRO task of the address space specified by pause_element_owner_stoken terminates. If the PE is being used to pause a DU when the CMRO task terminates, the system will release the DU using the owner_termination_release_code before the PE is deallocated. Note that in this case, the UPET returned will be 16 bytes of binary zeros, an invalid value.</td>
</tr>
<tr>
<td></td>
<td>• The PE is being used to pause a task or SRB which is asynchronously terminated via CALLRTM TYPE=ABTERM (for example, cancel or detach) or a PURGEDQ</td>
</tr>
<tr>
<td></td>
<td>• The PE is being used to pause a task or SRB when the home address space of the task or SRB is terminated</td>
</tr>
<tr>
<td></td>
<td>• The CMRO task of the home address space of the task or SRB which last used the PE terminates and the PE is not being used to pause an SRB</td>
</tr>
<tr>
<td></td>
<td>The home address space of the task or SRB which last used the PE terminates. Note: A PE is considered as &quot;being used to pause a task or SRB,&quot; when the PE is not Reset or Prereleased.</td>
</tr>
</tbody>
</table>

owner_termination_release_code
Supplied parameter
• Type: Character string
• Character Set: N/A
• Length: 3 bytes

Specifies the release code which will be returned to a paused DU if the system deallocates the pause element while it is being used to pause a task or SRB, due to the CMRO task of its owning address space terminating.

Note: If the system deallocates a PE due to its owner terminating while the PE was not being used to pause a task or SRB, future attempts to use the PE will fail with a return code indicating the PETOKEN was stale or the PE is in an invalid state.

linkage
Supplied parameter
• Type: Integer
• Character Set: N/A
• Length: 4 bytes

Specifies how the Allocate_Pause_Element service routine is to be invoked. The following options are supported:
IEA4APE2 callable service

Table 66. Linkage option

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_SVC</td>
<td>0</td>
<td>The Allocate_Pause_Element service routine will be invoked via an SVC linkage. This option can be used when in non-cross memory task mode, any key, and either problem state or supervisor state.</td>
</tr>
<tr>
<td>IEA_LINKAGE_BRANCH</td>
<td>1</td>
<td>The Allocate_Pause_Element service routine will be invoked via a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.</td>
</tr>
</tbody>
</table>

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and the return_code parameter contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (0)</td>
<td>IEA_SUCCESS</td>
<td><strong>Meaning:</strong> Successful completion. Action: None.</td>
</tr>
<tr>
<td>24 (18)</td>
<td>IEA_LOCK_HELD</td>
<td><strong>Meaning:</strong> Program error. One or more locks other than the local lock are held. The system rejects the service call. Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>IEA_UNSUPPORTED_MVSRELEASE</td>
<td><strong>Meaning:</strong> Environmental error. The system release does not support this service. The system rejects the service call. Action: Run the program on a system that supports the service.</td>
</tr>
<tr>
<td>40 (28)</td>
<td>IEA_INVALID_AUTHCODE</td>
<td><strong>Meaning:</strong> Program error. The pause_element_auth_level value specified in the call is not valid. The system rejects the service call. Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>IEA_INVALID_MODE</td>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
</tbody>
</table>
## IEA4APE2 callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 48 (30)                      | IEA_OUT_OF_STORAGE             | **Meaning:** Environmental error. The system could not obtain storage for a pause element. The system rejects the service call.  
**Action:** Retry the request later. If the problem persists, consult your system programmer. |
| 56 (38)                      | IEA_NO_PETS_AVAILABLE          | **Meaning:** There are no pause element tokens available.  
**Action:** Try the request again later.                                |
| 4095 (FFF)                   | IEA_UNEXPECTED_ERROR           | **Meaning:** This service routine encountered an unexpected error. The system rejects this service request.  
**Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
IEA4APE2 callable service
Chapter 63. IEA4DPE - Deallocate_Pause_Element

Description

Deallocate_Pause_Element frees a pause element that is no longer needed.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN</td>
</tr>
<tr>
<td></td>
<td>• For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN</td>
</tr>
</tbody>
</table>

AMODE: 64-bit
ASC mode: Primary mode.
Interrupt status: Enabled for I/O and external interrupts
Locks:
• When supervisor state and PSW key 0: The local lock may be held.
• When problem state, or not PSW key 0: No locks may be held.

Control parameters: Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling Deallocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
</tbody>
</table>
IEA4DPE callable service

13 Address of a 144-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1 Used as work registers by the system</td>
</tr>
<tr>
<td>2-13 Unchanged</td>
</tr>
<tr>
<td>14 Used as a work register by the system</td>
</tr>
<tr>
<td>15 Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1 Used as work registers by the system</td>
</tr>
<tr>
<td>2-13 Unchanged</td>
</tr>
<tr>
<td>14-15 Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTATE AMODE64=YES</td>
<td></td>
</tr>
<tr>
<td>CALL IEA4DPE</td>
<td>,(return_code, auth_level, pause_element_token)</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

return_code

Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Deallocate_Pause_Element service.
IEA4DPE callable service

auth_level
Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum authorization level of the pause element being deallocated. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (HEX)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>This pause element being deallocated must have been allocated with auth_level=IEA_UNAUTHORIZED.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>This pause element being deallocated must have been allocated with auth_level=IEA_AUTHORIZED.</td>
</tr>
</tbody>
</table>

pause_element_token
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element that is no longer needed.

ABEND codes
None.

Return codes
When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return code in: Dec (Hex)</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>Meaning: Successful completion.</td>
</tr>
<tr>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>IEA_SUCCESS</td>
<td></td>
</tr>
<tr>
<td>04 (04)</td>
<td>Meaning: Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>Meaning: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET be returned on Pause or Transfer.</td>
</tr>
<tr>
<td></td>
<td>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
</tbody>
</table>
### IEA4DPE callable service

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
</tr>
<tr>
<td>Equate symbol</td>
<td></td>
</tr>
</tbody>
</table>
| 24 (18)         | **IEA_LOCK_HELD**  | **Meaning:** Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 32 (20)         | **IEA_PE_BAD_STATE** | **Meaning:** Program error. The pause element associated with the specified pause element token specified is invalid or has already been paused. A paused PE must be released before it is deallocated. This return code also can indicate that the address space associated with the pause element is ending or has ended and that the system freed the pause element.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 36 (24)         | **IEA_UNSUPPORTED_MVS_RELEASE** | **Meaning:** Environmental error. The system release does not support this service. The system rejects the service call.  
**Action:** Run the program on a system that supports the service. |
| 40 (28)         | **IEA_INVALID_AUTHCODE** | **Meaning:** Program error. The auth_level value specified in the call is not valid. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 44 (2C)         | **IEA_INVALID_MODE** | **Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 60 (3C)         | **IEA_AUTH_TOKEN** | **Meaning:** Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call.  
**Action:** Program error. The specified pause element token is not valid. The system rejects the service call. |
| 64 (40)         | **IEA_PE_NOT_HOME** | **Meaning:** Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 4095 (FFF)      | **IEA_UNEXPECTED_ERROR** | **Meaning:** This service routine encountered an unexpected error. The system rejects this service request.  
**Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
Chapter 64. IEA4DPE2 — Deallocate_Pause_Element

Description
Deallocate_Pause_Element frees a pause element that is no longer needed.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>• For LINKAGE=BRANCH, supervisor state and PSW key 0.</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC:</td>
</tr>
<tr>
<td></td>
<td>– Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7.</td>
</tr>
<tr>
<td></td>
<td>– Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.</td>
</tr>
</tbody>
</table>

| Dispatchable unit mode: | • For LINKAGE=BRANCH: Task or SRB |
|                        | • For LINKAGE=SVC: Task |
| Cross memory mode:     | • For LINKAGE=BRANCH: Any PASN, any HASN, any SASN |
|                       | • For LINKAGE=SVC: PASN=HASN=SASN |
| AMODE:                | 64-bit |
| ASC mode:             | Primary mode. |
| Interrupt status:     | Enabled for I/O and external interrupts |
| Locks:                | • For LINKAGE=BRANCH: The local lock may be held. |
|                       | • For LINKAGE=SVC: No locks may be held. |
| Control parameters:   | Must in the primary address space and addressable by the caller. |

Programming requirements
Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB) or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions
Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Input register information
Before calling Deallocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:
IEA4DPE2 callable service

Register

Contents

1 Address of the parameter address list.
13 Address of a 144-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register

Contents

0-1 Used as work registers by the system
2-13 Unchanged
14 Used as a work register by the system
15 Return code

When control returns to the caller, the access registers (ARs) contain:

Register

Contents

0-1 Used as work registers by the system
2-13 Unchanged
14-15 Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTATE AMODE64=YES</td>
<td></td>
</tr>
<tr>
<td>CALL IEA4DPE2</td>
<td>/(return_code</td>
</tr>
<tr>
<td></td>
<td>/pause_element_token</td>
</tr>
<tr>
<td></td>
<td>/linkage)</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

return_code

Returned parameter

• Type: Integer
IEA4DPE2 callable service

- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Deallocate_Pause_Element service.

\textit{\texttt{pause\_element\_token}}

Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element that is no longer needed.

\textit{\texttt{linkage}}

Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Deallocate_Pause_Element service routine is to be invoked.
The following options are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_SVC</td>
<td>0</td>
<td>The Deallocate_Pause_Element service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and in either problem state or supervisor state.</td>
</tr>
<tr>
<td>IEA_LINKAGE_BRANCH</td>
<td>1</td>
<td>The Deallocate_Pause_Element service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.</td>
</tr>
</tbody>
</table>

\textbf{ABEND codes}

None.

\textbf{Return codes}

When the service returns control to the resource manager, GPR 15 and the \texttt{return\_code} parameter contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>IEA_SUCCESS</td>
<td>\textbf{Meaning: Successful completion} \nAction: None.</td>
</tr>
<tr>
<td>04 (04)</td>
<td>IEA_PE_TOKEN_BAD</td>
<td>\textbf{Meaning: Program error. The specified pause element token is not valid. The system rejects the service call.} \nAction: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
</tbody>
</table>
IEA4DPE2 callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08 (08)                       | IEA_PE_TOKEN_STALE         | **Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 24 (18)                       | IEA_LOCK_HELD              | **Meaning:** Program error. One or more locks other than the local lock are held. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 32 (20)                       | IEA_PE_BAD_STATE           | **Meaning:** Program error. The pause element associated with the specified pause element token is invalid or has already been paused. A paused PE must be released before it is deallocated. This return code also can indicate that the address space associated with the pause element is ending or has ended and that the system freed the pause element.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 36 (24)                       | IEA_UNSUPPORTED_MVS_RELEASE | **Meaning:** Environmental error. The system release does not support this service. The system rejects the service call.  
**Action:** Run the program on a system that supports the service. |
| 44 (2C)                       | IEA_INVALID_MODE           | **Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 64 (40)                       | IEA_PE_NOT_HOME            | **Meaning:** Program error. The pause element token was for an unauthorized pause element allocated to another address space.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 4095 (FFF)                    | IEA_UNEXPECTED_ERROR       | **Meaning:** This service routine encountered an unexpected error. The system rejects this service request.  
**Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
Chapter 65. IEA4PSE — Pause service

Description

Call IEA4PSE service to make the current task nondispatchable. After you pause a task, it remains nondispatchable until a release service specifying the same PET is called. That is, the program issuing the pause does not receive control back until after the release occurs.

If a release service specifying the same PET is called before pause, the system returns control immediately to the calling program, and the task is not paused.

When you use pause, it returns an updated PET. Use this updated PET to either deallocate or reuse the PE.

Environment

The requirements for the caller are:

Environmental factor | Requirement
---|---
Minimum authorization: Problem state and any PSW key.
Dispatchable unit mode:
  - When supervisor state and PSW key 0: Task or SRB.
  - When problem state, or not PSW key 0: Task.
Cross memory mode:
  - For auth_level=IEA_UNAUTHORIZED:
    PASN=HASN=SASN
  - For auth_level=IEAAUTHORIZED: Any PASN, any HASN, any SASN
AMODE: 64-bit
ASC mode: Primary mode.
Interrupt status: Enabled for I/O and external interrupts
Locks: No locks held.
Control parameters: Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

When the calling program is running auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only pause another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.
IE4PSE callable service

**Input register information**
Before calling the Pause service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 144-byte register save area.</td>
</tr>
</tbody>
</table>

**Output register information**
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**
None.

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTATE AMODE64=YES</td>
<td></td>
</tr>
<tr>
<td>CALL IE4PSE</td>
<td>/return_code</td>
</tr>
<tr>
<td></td>
<td>,auth_level</td>
</tr>
<tr>
<td></td>
<td>,pause_element_token</td>
</tr>
<tr>
<td></td>
<td>,updated_pause_element_token</td>
</tr>
<tr>
<td></td>
<td>,release_code</td>
</tr>
</tbody>
</table>

**Parameters**
The parameters are explained as follows:
IEA4PSE callable service

**return_code**
Returned parameter
- **Type:** Integer
- **Character Set:** N/A
- **Length:** 4 bytes

Contains the return code from the Pause service.

**auth_level**
Supplied parameter
- **Type:** Integer
- **Character Set:** N/A
- **Length:** 4 bytes

Indicates the maximum level that the specified pause element was allocated with. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (HEX)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>The pause element being paused must have been allocated with auth_level=IEA_UNAUTHORIZED.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>The pause element being paused must have been allocated with auth_level=IEA_AUTHORIZED.</td>
</tr>
</tbody>
</table>

**pause_element_token**
Supplied parameter
- **Type:** Character string
- **Character Set:** N/A
- **Length:** 16 bytes

A pause element token that identifies the pause element being used to pause the current task. You can obtain the PET from the Allocate_Pause_Element service.

When you use a PET in a call to the pause service, you cannot reuse the PET on a second call to pause or on a call to transfer. The pause service returns a new PET in updated_pause_element_token. The new PET now identifies the pause element used to pause the task; use the new PET the next time when you make a pause request using the same pause element.

**updated_pause_element_token**
Returned parameter
- **Type:** Character string
- **Character Set:** N/A
- **Length:** 16 bytes

A new pause element token that identifies the pause element originally identified by the PET specified in pause_element_token, which cannot be reused after a successful call to Pause.

**release_code**
Returned parameter
- **Type:** Character string
- **Character Set:** N/A
IEA4PSE callable service

- Length: 3 bytes
  
  The release code, specified by the issuer of the Release service. A Release that specified this code released the task from its paused condition.

ABEND codes

<table>
<thead>
<tr>
<th>Abend Code</th>
<th>Reason Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC7</td>
<td>001A0001</td>
<td>This is an internal error. Contact IBM support.</td>
</tr>
</tbody>
</table>

Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
</tr>
<tr>
<td>Equate symbol</td>
<td></td>
</tr>
</tbody>
</table>
| 00 (00)         | Meaning: Successful completion.  
Action: None. |
| IEA_SUCCESS     |                    |
| 04 (04)         | Meaning: Program error. The specified pause element token is not valid. The system rejects the service call.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| 08 (08)         | Meaning: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET be returned on Pause or Transfer.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| IEA_PE_TOKEN_STALE |                    |
| 12 (0C)         | Meaning: The work unit has already been paused using the specified pause element token. The system rejects the service call.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| IEA_DUPLICATE_PAUSE |                    |
| 24 (18)         | Meaning: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| IEA_LOCK_HELD   |                    |
| 32 (20)         | Meaning: Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call.  
Action: Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it. |
| IEA_PE_BAD_STATE |                    |
| 36 (24)         | Meaning: Environmental error. The system release does not support this service. The system rejects the service call.  
Action: Run the program on a system that supports the service. |
## IEA4PSE callable service

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
</tr>
<tr>
<td>Equate symbol</td>
<td></td>
</tr>
</tbody>
</table>

### IEA4PSE Return Codes and Meanings

<table>
<thead>
<tr>
<th>Return Code</th>
<th>IEA4PSE Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 (28)</td>
<td>IEA_INVALID_AUTHCODE</td>
<td>Program error. The auth_level value specified in the call is not valid. The system rejects the service call.</td>
<td>Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>IEA_INVALID_MODE</td>
<td>Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.</td>
<td>Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>52 (34)</td>
<td>IEA_ALREADY_SUSPENDED</td>
<td>The pause element was already paused.</td>
<td>Check the calling program for a probable coding error and correct the program and rerun it.</td>
</tr>
<tr>
<td>60 (3C)</td>
<td>IEA_AUTH_TOKEN</td>
<td>Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call.</td>
<td>Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td>64 (40)</td>
<td>IEA_PE_NOT_HOME</td>
<td>Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address.</td>
<td>Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>4095 (FFF)</td>
<td>IEA_UNEXPECTED_ERROR</td>
<td>This service routine encountered an unexpected error. The system rejects this service request.</td>
<td>Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</td>
</tr>
</tbody>
</table>
IEA4PSE callable service
Chapter 66. IEA4PSE2 — Pause service

Description
Call IEA4PSE2 service to make the current task or SRB nondispatchable. After you pause a task or SRB, it remains nondispatchable until a release or transfer specifying the same PET is called. That is, the program issuing the pause does not receive control back until after the RELEASE or TRANSFER occurs. At that time, the returned release_code contains a value supplied by the associated release or transfer request.

If a release service specifying the same PET is called before pause, the system returns control immediately to the calling program, and the task or SRB is not paused.

When you use pause, it returns an updated PET. Use this updated PET to either deallocate or reuse the PE.

Environment
The requirements for the caller are:

Environmental factor Requirement
Minimum authorization:  
- For LINKAGE=BRANCH, supervisor state and PSW key 0.
- For LINKAGE=SVC:  
  - Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7.
  - Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:  
- For LINKAGE=BRANCH: Task or SRB
- For LINKAGE=SVC: Task
Cross memory mode:  
- For LINKAGE=BRANCH: Any PASN, any HASN, any SASN
- For LINKAGE=SVC: PASN=HASN=SASN
AMODE:  
- 64-bit
ASC mode:  
- Primary mode.
Interrupt status:  
- Enabled for I/O and external interrupts.
Locks:  
- No locks held.
Control parameters:  
- Must be in the primary address space and addressable by the caller.

Programming requirements
Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB) or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>
IEA4PSE2 callable service

Restrictions
Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Pause cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information
Before calling the Pause service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 144-byte register save area.</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTATE AMODE64=YES</td>
<td></td>
</tr>
</tbody>
</table>
IEA4PSE2 callable service

## Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL IEA4PSE2</td>
<td></td>
</tr>
</tbody>
</table>

{return_code
,pause_element_token
,updated_pause_element_token
,release_code
,linkage)

## Parameters

The parameters are explained as follows:

**return_code**
- Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes
- Contains the return code from the Pause service.

**pause_element_token**
- Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes
- A pause element token that identifies the pause element being used to pause the current task or SRB. You obtain the PET from the Allocate_Pause_Element service.

When you use a PET in a call to the pause service, you cannot reuse the PET on a second call to pause or on a call to Transfer. The pause service returns a new PET in updated_pause_element_token. The new PET now identifies the pause element used to pause the task or SRB; use the new PET the next time when you make a Pause request using the same pause element.

**updated_pause_element_token**
- Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes
- A new pause element token that identifies the pause element originally identified by the PET specified in pause_element_token. This new PET must be used in place of the PET specified in pause_element_token on future calls to the Pause, Release, Transfer, or Deallocate_Pause_Element service.

**release_code**
- Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes
- The release code is specified by the issuer of the release service, which can release the task or SRB of the paused condition.
IEA4PSE2 callable service

`linkage`
Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Pause service routine is to be invoked. The following options are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_SVC</td>
<td>0</td>
<td>The Pause service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and either problem state or supervisor state.</td>
</tr>
<tr>
<td>IEA_LINKAGE_BRANCH</td>
<td>1</td>
<td>The Pause service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.</td>
</tr>
</tbody>
</table>

ABEND codes

<table>
<thead>
<tr>
<th>Abend Code</th>
<th>Reason Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC7</td>
<td>001A0001</td>
<td>This is an internal error. Contact IBM support.</td>
</tr>
</tbody>
</table>

Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>IEA_SUCCESS</td>
<td>Meaning: Successful completion. Action: None</td>
</tr>
<tr>
<td>04 (04)</td>
<td>IEA_PE_TOKEN_BAD</td>
<td>Meaning: Program error. The specified pause element token is not valid. The system rejects the service call. Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>IEA_PE_TOKEN_STALE</td>
<td>Meaning: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>12 (0C)</td>
<td>IEA_DUPLICATE_PAUSE</td>
<td>Meaning: The work unit has already been paused using the specified pause element token. The system rejects the service call. Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
</tbody>
</table>
### IEA4PSE2 callable service

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24 (18)</td>
<td>IEA_LOCK_HELD</td>
<td><strong>Meaning:</strong> Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>32 (20)</td>
<td>IEA_PE_BAD_STATE</td>
<td><strong>Meaning:</strong> Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>IEA_UNSUPPORTED_MVS_RELEASE</td>
<td><strong>Meaning:</strong> Environmental error. The system release does not support this service. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Run the program on a system that supports the service.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>IEA_INVALID_MODE</td>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>52 (34)</td>
<td>IEA_ALREADY_SUSPENDED</td>
<td><strong>Meaning:</strong> The pause element was already paused.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error and correct the program and rerun it.</td>
</tr>
<tr>
<td>60 (3C)</td>
<td>IEA_AUTH_TOKEN</td>
<td><strong>Meaning:</strong> Program error. The caller was in Problem state or key 8, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td>64 (40)</td>
<td>IEA_PE_NOT_HOME</td>
<td><strong>Meaning:</strong> Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>76 (4C)</td>
<td>IEA_ABENDED_47B</td>
<td><strong>Meaning:</strong> After an SRB received ABEND 47B, it invoked IEA4PSE2. It is not valid to invoke IEA4PSE2 after receiving ABEND 47B.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Update the calling program to not invoke IEA4PSE2 after ABEND 47B.</td>
</tr>
<tr>
<td>80 (50)</td>
<td>IEA_IN_SUSPEND_EXIT</td>
<td><strong>Meaning:</strong> The suspend exit specified on SUSPEND with SPTOKEN of an SRB invoked IEA4PSE2. It is not valid to invoke IEA4PSE2 from a suspend exit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Update the calling program to not invoke IEA4PSE2 from a suspend exit.</td>
</tr>
</tbody>
</table>
### IEA4PSE2 callable service

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 4095 (FFF)      | IEA_UNEXPECTED_ERROR | **Meaning:** This service routine encountered an unexpected error. The system rejects this service request.  
**Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
Chapter 67. IEA4RLS — Release

Description

Call IEA4RLS service to remove a task that has been paused, or to keep a task from being paused. Although a pause element can be used multiple times to pause a task, a pause element token can be used to successfully pause and release a task only once. Each time a pause element is used, the system generates a new PET to identify the pause element. The system returns the new updated PET on calls to the pause and transfer services.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>• When supervisor state and PSW key 0: Task or SRB.</td>
</tr>
<tr>
<td></td>
<td>• When problem state, or not PSW key 0: Task.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>• For auth_level=IEA_UNAUTHORIZED:</td>
</tr>
<tr>
<td></td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td></td>
<td>• For auth_level=IEA_AUTHORIZED:</td>
</tr>
<tr>
<td></td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>May hold the CPU, local or CMS lock.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the calling program’s object code with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling the Release service, the caller must ensure that the following general purpose (GPRs) contain the specified information:
IEA4RLS callable service

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 144-byte register save area.</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSSTATE AMODE64=YES</td>
<td></td>
</tr>
<tr>
<td>CALL IEA4RLS</td>
<td>./return_code</td>
</tr>
<tr>
<td></td>
<td>./auth_level</td>
</tr>
<tr>
<td></td>
<td>./target_du_pause_element_token</td>
</tr>
<tr>
<td></td>
<td>./target_du_release_code</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

return_code
  Returned parameter
  - Type: Integer
  - Character Set: N/A
IEA4RLS callable service

- Length: 4 bytes
  Contains the return from the Release service.

auth_level
Supplied Parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum authorization level that the specified pause element was allocated with. The calling program can use constants IEA_UNAUTHORIZED and IEAAUTHORIZED, defined by IEAASM and IEAC. The following levels are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (HEX)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>The pause element being released must have been allocated with auth_level=IEA_UNAUTHORIZED.</td>
</tr>
<tr>
<td>IEAAUTHORIZED</td>
<td>1</td>
<td>The pause element being released must have been allocated with auth_level=IEAAUTHORIZED.</td>
</tr>
</tbody>
</table>

target_du_pause_element_token
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element used to pause the task. If the PET identifies a pause element that has not been paused, the task is paused. However, the value specified in target_du_release_code is returned to the caller of pause.

target_du_release_code
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the caller of pause or transfer service that used or will use the same PET to pause a task. If your program is not using this code for communication, set this field to zero.

ABEND codes
None.

Return codes
When the service returns control to the resource manager, GPR 15 and the return_code parameter contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>Meaning: Successful completion.</td>
</tr>
<tr>
<td>IEA_SUCCESS</td>
<td>Action: None.</td>
</tr>
</tbody>
</table>
### IEA4RLS callable service

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decimal (Hex)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Equate symbol</strong></td>
<td></td>
</tr>
<tr>
<td>04 (04)</td>
<td><strong>IEA_PE_TOKEN_BAD</strong></td>
</tr>
<tr>
<td><strong>Meaning:</strong> The specified pause element token is not valid. The system rejects the service call.</td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>08 (08)</td>
<td><strong>IEA_PE_TOKEN_STALE</strong></td>
</tr>
<tr>
<td><strong>Meaning:</strong> The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET be returned on Pause or Transfer.</td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>16 (10)</td>
<td><strong>IEA_SLEEP_DISRUPTED</strong></td>
</tr>
<tr>
<td><strong>Meaning:</strong> RTM has ended the task; no release is necessary.</td>
<td><strong>Action:</strong> None.</td>
</tr>
<tr>
<td>20 (14)</td>
<td><strong>IEA_SPACE_TERMINATING</strong></td>
</tr>
<tr>
<td><strong>Meaning:</strong> The address space that contains the task is terminating; no release is necessary.</td>
<td><strong>Action:</strong> None.</td>
</tr>
<tr>
<td>24 (18)</td>
<td><strong>IEA_LOCK_HELD</strong></td>
</tr>
<tr>
<td><strong>Meaning:</strong> Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call.</td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>32 (20)</td>
<td><strong>IEA_PE_BAD_STATE</strong></td>
</tr>
<tr>
<td><strong>Meaning:</strong> Program error. The pause element associated with the pause element token specified is not valid or has already been prereleased.</td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>36 (24)</td>
<td><strong>IEA_UNSUPPORTED_MVS_RELEASE</strong></td>
</tr>
<tr>
<td><strong>Meaning:</strong> Environmental error. The system release does not support this service. The system rejects the service call.</td>
<td><strong>Action:</strong> Run the program on a system that supports the service.</td>
</tr>
<tr>
<td>40 (28)</td>
<td><strong>IEA_INVALID_AUTHCODE</strong></td>
</tr>
<tr>
<td><strong>Meaning:</strong> Program error. The auth_level value specified in the call is not valid. The system rejects the service call.</td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td><strong>IEA_INVALID_MODE</strong></td>
</tr>
<tr>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.</td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>60 (3C)</td>
<td><strong>IEA_AUTH_TOKEN</strong></td>
</tr>
<tr>
<td><strong>Meaning:</strong> Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call.</td>
<td><strong>Action:</strong> Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
</tbody>
</table>
## IEA4RLS callable service

<table>
<thead>
<tr>
<th>Return code in:</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
</tr>
<tr>
<td>Equate symbol</td>
<td></td>
</tr>
<tr>
<td>64 (40)</td>
<td><strong>IEA_PE_NOT_HOME</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>4095 (FFF)</td>
<td><strong>IEA_UNEXPECTED_ERROR</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: This service routine encountered an unexpected error. The system rejects this service request.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Contact IBM support.</td>
</tr>
</tbody>
</table>
IEA4RLS callable service
Chapter 68. IEA4RLS2 — Release

Description

Call IEA4RLS2 service to remove a task or SRB that has been paused, or to keep a task or SRB from being paused.

Although a pause element can be used multiple times to pause a task or SRB, a pause element token can be used to successfully pause and release a task or SRB only once. Each time a pause element is used, the system generates a new PET to identify the pause element. The system returns the new updated PET on calls to the pause and transfer services.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>• For LINKAGE=BRANCH, supervisor state and PSW key 0.</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC:</td>
</tr>
<tr>
<td></td>
<td>- Working with an IEA.AUTHORIZED pause element, supervisor state and PSW key 0-7.</td>
</tr>
<tr>
<td></td>
<td>- Working with an IEA.UNAUTHORIZED pause element, problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>• For LINKAGE=BRANCH: Task or SRB</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC: Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC: PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>• For LINKAGE=BRANCH: The CPU, CMS, or local locks may be held.</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC: No locks may be held.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB) or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>
IEA4RLS2 callable service

Restrictions
Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Release cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information
Before calling the Release service, the caller must ensure that the following general purpose (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 144-byte register save area.</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTATE AMODE64=YES</td>
<td></td>
</tr>
</tbody>
</table>
### Parameters

The parameters are explained as follows:

- **return_code**
  - Returned parameter
  - Type: Integer
  - Character Set: N/A
  - Length: 4 bytes
  - Contains the return from the Release service.

- **target_du_pause_element_token**
  - Supplied parameter
  - Type: Character string
  - Character Set: N/A
  - Length: 16 bytes
  - Contains the pause element token that identifies the pause element used to pause a task or SRB. If the PET identifies a pause element that has not been paused, the task is paused. However, the value specified in target_du_release_code is returned to the caller of pause.

- **target_du_release_code**
  - Supplied parameter
  - Type: Character string
  - Character Set: N/A
  - Length: 3 bytes
  - Contains the release code returned to the caller of pause or transfer service that used or will use the same PET to pause a task or SRB. If the program is not using this code for communication, set this field to zero.

- **linkage**
  - Supplied parameter
  - Type: Integer
  - Character Set: N/A
  - Length: 4 bytes
  - Specifies how the Release service routine is to be invoked. The following options are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_SVC</td>
<td>0</td>
<td>The Release service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and either problem state or supervisor state.</td>
</tr>
</tbody>
</table>
### IEA4RLS2 callable service

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_BRANCH</td>
<td>1</td>
<td>The Release service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.</td>
</tr>
</tbody>
</table>

### ABEND codes

None.

### Return codes

When the service returns control to the resource manager, GPR 15 and the `return_code` parameter contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00  (00)    | IEA_SUCCESS            | Meaning: Successful completion.  
Action: None.                               |
| 04  (04)    | IEA_PE_TOKEN_BAD      | Meaning: The specified pause element token is not valid. The system rejects the service call.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| 08  (08)    | IEA_PE_TOKEN_STALE    | Meaning: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| 16  (10)    | IEA_SLEEP_DISRUPTED    | Meaning: RTM has terminated the task or SRB; no release is necessary.  
Action: None.                               |
| 20  (14)    | IEA_SPACE_TERMINATING | Meaning: The address space that contains the task or SRB is terminating; no release is necessary.  
Action: None.                               |
| 24  (18)    | IEA_LOCK_HELD         | Meaning: Program error. The caller is holding one or more locks; other than the local lock, CMS, or CPU lock, no locks may be held. The system rejects the service call.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| 32  (20)    | IEA_PE_BAD_STATE      | Meaning: Program error. The pause element associated with the pause element token specified is invalid or has already been prereleased.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| 36  (24)    | IEA_UNSUPPORTED_MVS_RELEASE | Meaning: Environmental error. The system release does not support this service. The system rejects the service call.  
Action: Run the program on a system that supports the service. |
<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>44 (2C)</td>
<td>IEA_INVALID_MODE</td>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>60 (3C)</td>
<td>IEA_AUTH_TOKEN</td>
<td><strong>Meaning:</strong> Program error. The caller was in Problem state or key 8, but the pause element token was allocated with pause_element_auth_level=IEAAUTHORIZED. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td>64 (40)</td>
<td>IEA_PE_NOT_HOME</td>
<td><strong>Meaning:</strong> Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>4095 (FFF)</td>
<td>IEA_UNEXPECTED_ERROR</td>
<td><strong>Meaning:</strong> This service routine encountered an unexpected error. The system rejects this service request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</td>
</tr>
</tbody>
</table>
IEA4RLS2 callable service
Chapter 69. IEA4RPI — Retrieve_Pause_Element_Information service

Description

Call Retrieve_Pause_Element_Information to get information about a pause element. The information returned includes:

- The authorization level of the pause element
- The address space that currently owns the pause element
- The current state (reset, prereleased, paused, or released) of the pause element
- If the state of the pause element is prereleased or released, the release code of the pause element

An authorized program can use Retrieve_Pause_Element_Information to test the validity of a pause element passed by an unauthorized program. The authorized program can do this to ensure that it does not perform any operation, such as releasing the pause element, unless the unauthorized program is also able to perform the same operation.

Environment

The requirements for the caller are:

Environmental factor | Requirement
--- | ---
Minimum authorization | Problem state and any PSW key.
Dispatchable unit mode | 
- When supervisor state and PSW key 0: Task or SRB.
- When problem state, or not PSW key 0: Task.
Cross memory mode | 
- When supervisor state and PSW key 0: Any PASN, any HASN, any SASN
- When problem state, or not PSW key 0: PASN=HASN=SASN
AMODE: | 64-bit
ASC mode: | Primary mode.
Interrupt status: | Enabled for I/O and external interrupts.
Locks: | No locks held.
Control parameters: | Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

None.
IEA4RPI callable service

Input register information
Before calling the Retrieve_Pause_Element_Information service, the caller does not need to place any information into any register, unless using it in register notation for the parameters, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEST AMODE64=YES</td>
<td></td>
</tr>
<tr>
<td>CALL IEA4RPI</td>
<td>{return_code, auth_level, pause_element_token, authorization, owner, state, release_code}</td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained as follows:

return_code
- Returned parameter
- Type: Integer
IEA4RPI callable service

Contains the return code from the Retrieve_Pause_Element_Information service.

 auth_level
 Returned parameter
 Type: Integer
 Character Set: N/A
 Length: 4 bytes

Indicates the caller's authorization level. The following levels are supported:
IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which can be used by the calling program.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>The caller is not key 0 and supervisor state.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>The caller is both key 0 and supervisor state.</td>
</tr>
</tbody>
</table>

 pause_element_token
 Supplied parameter
 Type: Character string
 Character Set: N/A
 Length: 16 bytes

A pause element token that identifies the pause element for which information will be returned. You can obtain the PET from the Allocate_Pause_Element service.

 authorization
 Returned parameter
 Type: Integer
 Character Set: N/A
 Length: 4 bytes

The authorization level of the creator of the pause element specified by the input PET.

One of the following values:

<table>
<thead>
<tr>
<th>IEAASM and IEAC defined constants</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>The caller is not key 0 and supervisor state.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>The caller is not key 0 and supervisor state.</td>
</tr>
<tr>
<td>IEA_UNAUTHORIZED + IEA_CHECKPOINTOK</td>
<td>2</td>
<td>Unauthorized PET that can tolerate the pause elements' not being restored upon a restart after a checkpoint.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED + IEA_CHECKPOINTOK</td>
<td>3</td>
<td>Authorized PET that can tolerate the pause elements' not being restored upon a restart after a checkpoint.</td>
</tr>
</tbody>
</table>

 owner
 Returned parameter
 Type: Character string
IEA4RPI callable service

- Character Set: N/A
- Length: 8 bytes

The Stoken of the address space that currently owns the pause element specified by the input PET.

\texttt{\textbackslash \textasciitilde state}

Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The state of the pause element specified by the input PET.

**Note:** The value returned is the state at the time the service obtained it. The state might have changed after it was obtained.

<table>
<thead>
<tr>
<th>State Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA4_PET_PRERELEASED</td>
<td>The PE was released before any task or SRB was suspended on it, and no task or SRB has attempted to pause it.</td>
</tr>
<tr>
<td>1 (1)</td>
<td></td>
</tr>
<tr>
<td>IEA4_PET_RESET</td>
<td>The PE is not being used to make any task or SRB nondispatchable. If the PE is used in an attempt to pause the current task or SRB, the task or SRB will be made nondispatchable.</td>
</tr>
<tr>
<td>2 (2)</td>
<td></td>
</tr>
<tr>
<td>IEA4_PET_RELEASED</td>
<td>The task RB or SRB is currently dispatchable, but control has not been returned to the task or SRB following a call to the Pause or Transfer service.</td>
</tr>
<tr>
<td>40 (64)</td>
<td>A call to the Release or Transfer service has released the task or SRB. In either case, control has not been returned to the caller of the Pause or Transfer service. The system has not transited the PE into the RESET state.</td>
</tr>
<tr>
<td>IEA4_PET_PAUSED</td>
<td>A task RB or SRB is currently nondispatchable. Its dispatchability is controlled by the PE.</td>
</tr>
<tr>
<td>80 (128)</td>
<td></td>
</tr>
</tbody>
</table>

\texttt{\textbackslash \textasciitilde release\_code}

Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code is specified by the issuer of the release service, which can release the task or SRB from the paused condition.

**Note:** The returned value is random if the state parameter is not IEA4_PET_RELEASED or IEA4_PET_PRERELEASED.

**ABEND codes**

None.
### Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00 (00)                       | IEA_SUCCESS            | **Meaning:** Successful completion.  
Action: None.                                      |
| 04 (04)                       | IEA_PE_TOKEN_BAD       | **Meaning:** Program error. The specified pause element token is not valid. The system rejects the service call.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| 08 (08)                       | IEA_PE_TOKEN_STALE     | **Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| 24 (18)                       | IEA_LOCK_HELD          | **Meaning:** Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| 36 (24)                       | IEA_UNSUPPORTED_MVS_RELEASE | **Meaning:** Environmental error. The system release does not support this service. The system rejects the service call.  
Action: Run the program on a system that supports the service. |
| 44 (2C)                       | IEA_INVALID_MODE       | **Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
| 60 (3C)                       | IEA_AUTH_TOKEN         | **Meaning:** Program error. The caller specified an unauthorized auth_level type, but a pause element token allocated with an authorized auth_level type was encountered. The system rejects the service call.  
Action: Program error. The specified pause element token is not valid. The system rejects the service call. |
| 64 (40)                       | IEA_PE_NOT_HOME        | **Meaning:** Program error. The caller specified an unauthorized auth_level type, but a pause element token for a pause element allocated to another address space was specified.  
Action: Check the calling program for a probable coding error. Correct the program and rerun it. |
## IEA4RPI callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4095 (FFF)</td>
<td>IEA_UNEXPECTED_ERROR</td>
<td><strong>Meaning:</strong> This service routine encountered an unexpected error. The system rejects this service request. &lt;br&gt;<strong>Action:</strong> Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</td>
</tr>
</tbody>
</table>
Chapter 70. IEA4RPI2 — Retrieve_Pause_Element_Information service

Description

Call Retrieve_Pause_Element_Information to get information about a pause element. The information returned includes:

- The authorization level of the pause element
- The address space that currently owns the pause element
- The current state (reset, prereleased, paused, or released) of the pause element
- If the state of the pause element is prereleased or released, the release code of the pause element

An authorized program can use Retrieve_Pause_Element_Information to test the validity of a pause element passed by an unauthorized program. The authorized program can do this to ensure that it does not perform any operation, such as releasing the pause element, unless the unauthorized program is also able to perform the same operation.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=BRANCH, supervisor state and PSW key 0.</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC:</td>
</tr>
<tr>
<td></td>
<td>• Working with an IEAAUTHORIZED pause element, supervisor state and PSW key 0-7.</td>
</tr>
<tr>
<td></td>
<td>• Working with an IEAUNAUTHORIZED pause element, problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>• For LINKAGE=BRANCH: Task or SRB</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC:</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td></td>
<td>• For LINKAGE=SVC: PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the object code of calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:
IEA4RPI2 callable service

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

**Restrictions**

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

**Input register information**

Before calling the Retrieve_Pause_Element_Information service, the caller does not need to place any information into any register, unless using it in register notation for the parameters, or using it as a base register.

**Output register information**

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTATE AMODE64=YES</td>
<td></td>
</tr>
</tbody>
</table>
CALL IEA4RPI2

,return_code
,pause_element_token
,linkage
,owner
,current_stoken
,authorization
,state
,release_code)

Parameters
The parameters are explained as follows:

return_code
Returned parameter
• Type: Integer
• Character Set: N/A
• Length: 4 bytes
Contains the return code from the Retrieve_Pause_Element_Information service.

pause_element_token
Supplied parameter
• Type: Character string
• Character Set: N/A
• Length: 16 bytes
A pause element token that identifies the pause element for which information will be returned. You can obtain the PET from the Allocate_Pause_Element service.

linkage
Supplied parameter
• Type: Integer
• Character Set: N/A
• Length: 4 bytes
The calling program can use the constants defined in IEAASM or IEAC, as appropriate. Add the specified values together to achieve the desired results. For example, to specify linkage branch and untrusted PET, specify IEA_LINKAGE_BRANCH + IEA_UNTRUSTED_PET.

The following options are supported:

Table 67. Linkage variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_SVC</td>
<td>0</td>
<td>The Retrieve_Pause_Element_Information service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and in either problem state or supervisor state.</td>
</tr>
</tbody>
</table>
Table 67. Linkage variables (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_BRANCH</td>
<td>1</td>
<td>The Retrieve_Pause_Element_Information service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.</td>
</tr>
</tbody>
</table>

Table 68. Untrusted attribute linkage variable

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNTRUSTED_PET</td>
<td>2</td>
<td>The Retrieve_Pause_Element_Information service routine is to validate that the input PET is allowed to be used by an unauthorized program. An authorized program should use this if it is validating a PET provided to it by an unauthorized caller.</td>
</tr>
</tbody>
</table>

,owner

Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 8 bytes

The Stoken of the address space that currently owns the pause element specified by the input PET.

,current_stoken

Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 8 bytes

If the value returned in state is IEA_PET_PAUSED, The stoken of the home address space of the task or SRB which is paused by the specified pause element. If the value in state is not IEA_PET_PAUSED, the information returned in this parameter is undefined.

.authorization

Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The authorization level of the creator of the pause element specified by the input PET.

One of the following values:

<table>
<thead>
<tr>
<th>IEAASM and IEAC defined constants</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>The caller is not key 0 and supervisor state.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>The caller is not key 0 and supervisor state.</td>
</tr>
</tbody>
</table>
### IEA4RPI2 callable service

<table>
<thead>
<tr>
<th>IEAASM and IEAC defined constants</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED + IEA_CHECKPOINTOK</td>
<td>2</td>
<td>Unauthorized PET that can tolerate the pause elements' not being restored upon a restart after a checkpoint.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED + IEA_CHECKPOINTOK</td>
<td>3</td>
<td>Authorized PET that can tolerate the pause elements' not being restored upon a restart after a checkpoint.</td>
</tr>
</tbody>
</table>

**,state

Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The state of the pause element specified by the input PET.

**Note:** The value returned is the state at the time the service obtained it. The state might have changed after it was obtained.

<table>
<thead>
<tr>
<th>State Constant</th>
<th>Hexadecimal (Decimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA4_PET_PRERELEASE</td>
<td>1 (1)</td>
<td>The PE was released before any task or SRB was suspended on it, and no task or SRB has attempted to pause it.</td>
</tr>
<tr>
<td>IEA4_PET_RESET</td>
<td>2 (2)</td>
<td>The PE is not being used to make any task or SRB nondispatchable. If the PE is used in an attempt to pause the current task or SRB, the task or SRB will be made nondispatchable.</td>
</tr>
<tr>
<td>IEA4_PET_RELEASED</td>
<td>40 (64)</td>
<td>The task RB or SRB is currently dispatchable, but control has not been returned to the task or SRB following a call to the Pause or Transfer service. A call to the Release or Transfer service has released the task or SRB. In either case, control has not been returned to the caller of the Pause or Transfer service. The system has not transited the PE into the RESET state.</td>
</tr>
<tr>
<td>IEA4_PET_PAUSED</td>
<td>80 (128)</td>
<td>A task RB or SRB is currently nondispatchable. Its dispatchability is controlled by the PE.</td>
</tr>
</tbody>
</table>

**,release_code

Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code is specified by the issuer of the release service, which can release the task or SRB from the paused condition.

**Note:** The returned value is random if the state parameter is not IEA4_PET_RELEASED or IEA4_PET_PRERELEASED.

### ABEND codes

None.
### Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>IEA_SUCCESS</td>
<td><strong>Meaning:</strong> Successful completion.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None.</td>
</tr>
<tr>
<td>04 (04)</td>
<td>IEA_PE_TOKEN_BAD</td>
<td><strong>Meaning:</strong> Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>IEA_PE_TOKEN_STALE</td>
<td><strong>Meaning:</strong> The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>24 (18)</td>
<td>IEA_LOCK_HELD</td>
<td><strong>Meaning:</strong> Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>IEA_UNSUPPORTED_MVS_RELEASE</td>
<td><strong>Meaning:</strong> Environmental error. The system release does not support this service. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Run the program on a system that supports the service.</td>
</tr>
<tr>
<td>40 (28)</td>
<td>IEA_INVALID_AUTHCODE</td>
<td><strong>Meaning:</strong> Program error. The pause_element_auth_level value specified in the call is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>IEA_INVALID_MODE</td>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>60 (3C)</td>
<td>IEA_AUTH_TOKEN</td>
<td><strong>Meaning:</strong> Program error. The caller was in Problem state or key 8, or specified IEA_UNTRUSTED_PET, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td>64 (40)</td>
<td>IEA_PE_NOT_HOME</td>
<td><strong>Meaning:</strong> Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
</tbody>
</table>
### IEA4RPI2 callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 4095 (FFF)                    | IEA_UNEXPECTED_ERROR | **Meaning**: This service routine encountered an unexpected error. The system rejects this service request.  
**Action**: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
IEA4RPI2 callable service
Chapter 71. IEA4TPE — Test_Pause_Element service

Description

Call Test_Pause_Element to test a pause element and determine its state. If the state is prereleased or released, the release code of the pause element also is returned.

To ensure minimal overhead when you use the service, Test_Pause_Element establishes no recovery. You are responsible for supplying any needed recovery to handle errors that occur because of the incorrect input pause element tokens or call state errors.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>• When supervisor state and PSW key 0: Task or SRB.</td>
</tr>
<tr>
<td></td>
<td>• When problem state, or not PSW key 0: Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space and addressable by the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

None.

Input register information

Before calling the Test_Pause_Element service, the caller does not have to place any information into any register, unless using the input register in register notation for the parameters, or using the input register as a base register.

Output register information

When control returns to the caller, the GPRs contain:
IEA4TPE callable service

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1  Used as work registers by the system</td>
</tr>
<tr>
<td>2-13 Unchanged</td>
</tr>
<tr>
<td>14   Used as a work register by the system</td>
</tr>
<tr>
<td>15   Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1  Used as work registers by the system</td>
</tr>
<tr>
<td>2-13 Unchanged</td>
</tr>
<tr>
<td>14-15 Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSTATE AMODE64=YES</td>
<td></td>
</tr>
<tr>
<td>CALL IEA4TPE ,return_code</td>
<td>,pause_element_token ,state ,release_code</td>
</tr>
</tbody>
</table>
IEA4TPE callable service

A pause element token that identifies the pause element for which information is to be returned. You can obtain the PET from the Allocate_Pause_Element service.

, state
    Returned parameter
    • Type: Integer
    • Character Set: N/A
    • Length: 4 bytes

    The state of the pause element specified by the input PET.

    Note: The value returned is the state at the time the service obtained it. The state might have changed after it was obtained.

<table>
<thead>
<tr>
<th>State Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA4_PET_PRERELEASED 1 (1)</td>
<td>The PE was released before any task or SRB was suspended on it, and no task or SRB has attempted to pause it.</td>
</tr>
<tr>
<td>IEA4_PET_RESET 2 (2)</td>
<td>The PE is not being used to make any task or SRB nondispatchable. If the PE is used in an attempt to pause the current task or SRB, the task or SRB is made nondispatchable.</td>
</tr>
<tr>
<td>IEA4_PET_RELEASED 40 (64)</td>
<td>The task RB or SRB is currently dispatchable, but control has not been returned to the task or SRB following a call to the Pause or Transfer service. A call to the release or transfer service has released the task or SRB. In either case, control has not been returned to the caller of the pause or transfer service. The system has not change the PE into the RESET state.</td>
</tr>
<tr>
<td>IEA4_PET_PAUSED 80 (128)</td>
<td>A task RB or SRB is currently nondispatchable. Its dispatchability is controlled by the PE.</td>
</tr>
</tbody>
</table>

, release_code
    Returned parameter
    • Type: Character string
    • Character Set: N/A
    • Length: 3 bytes

    The release code is specified by the issuer of the Release service, which released the task or SRB from the paused condition.

    Note: The returned value is random if the state parameter is not IEA4_PET_RELEASED or IEA4_PET_PRERELEASED. 

ABEND codes
None.

Return codes
When the service returns control to your program, GPR 15 contains one of the following return codes:
### IEA4TPE callable service

<table>
<thead>
<tr>
<th>Return code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>IEA_SUCCESS</td>
<td>Meaning: Successful completion.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>04 (04)</td>
<td>IEA_PE_TOKEN_BAD</td>
<td>Meaning: Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>IEA_PE_TOKEN_STALE</td>
<td>Meaning: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
</tbody>
</table>
Chapter 72. IEA4XFR — Transfer service

Description

Call IEA4XFR service to release a paused task, and, when possible, give the task immediate control. This service can also, optionally, pause the task under which the transfer request is made. If the caller does not request that its task be paused, the caller’s task remains dispatchable.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>• When supervisor state and PSW key 0: Task or SRB.</td>
</tr>
<tr>
<td></td>
<td>• When problem state, or not PSW key 0: Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>• For auth_level=IEA_UNAUTHORIZED:</td>
</tr>
<tr>
<td></td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td></td>
<td>• For auth_level=IEA_AUTHORIZED:</td>
</tr>
<tr>
<td></td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
</tbody>
</table>

| AMODE                      | 64-bit                                           |
| ASC mode:                  | Primary mode.                                     |
| Interrupt status:          | Enabled for I/O and external interrupts.          |
| Locks:                     | When supervisor state and PSW key 0 and a current_du_pause_element_token of 16 bytes of binary zeros are specified, the local lock may be held. Otherwise, no locks may be held. |

Control parameters: Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only transfer to another task in its home address space. All pause element tokens (PETS) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling the Transfer service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:
IEA4XFR callable service

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 144-byte register save area.</td>
</tr>
</tbody>
</table>

**Output register information**

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSSTATE AMODE64=YES</td>
<td></td>
</tr>
</tbody>
</table>

CALL IEA4XFR

<table>
<thead>
<tr>
<th>return_code</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth_level</td>
</tr>
<tr>
<td>current_du_pause_element_token</td>
</tr>
<tr>
<td>updated_pause_element_token</td>
</tr>
<tr>
<td>current_du_release_code</td>
</tr>
<tr>
<td>target_du_pause_element_token</td>
</tr>
<tr>
<td>target_du_release_code</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

- **return_code**
  - Returned parameter
IEA4XFR callable service

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the transfer service.

auth_level
Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum authorization level of the pause element being deallocated. The calling program can use constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, defined by IEAASM and IEAC. The following levels are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (HEX)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_UNAUTHORIZED</td>
<td>0</td>
<td>The pause elements must have been allocated with auth_level=._UNAUTHORIZED.</td>
</tr>
<tr>
<td>IEA_AUTHORIZED</td>
<td>1</td>
<td>The pause elements must have been allocated with auth_level=_AUTHORIZED.</td>
</tr>
</tbody>
</table>

current_du_pause_element_token
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies the pause element used to pause the current task. When a PET is used on a call to the pause service, it cannot be reused on a second call to pause or as a current_du_pause_element_token on transfer. A new PET is returned to updated_pause_element_token. The new PET now properly defines the pause element and should be used the next time when a pause, transfer, release, or deallocate_pause_element request is using the same pause element.

If the value specified is 16-bytes of binary zeros, the current task will not be paused. The updated_pause_element_token and current_du_release_code are unpredictable.

CAUTION:
Do not specify the same PET for both current_du_pause_element_token and target_pause_element_token.

updated_pause_element_token
Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a new pause element token that identifies the pause element originally identified by the PET specified in current_du_pause_element_token. The PET originally specified in current_du_pause_element_token cannot be reused after a successful call to pause or transfer service.
IEA4XFR callable service

If you set the current_du_pause_element_token to zeros, the contents of updated_pause_element_token are unpredictable.

\texttt{current_du_release_code}

Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code set by the issuer of the release or transfer service that released the current task from the paused condition.

If you set the current_du_pause_element_token to zero, the contents are unpredictable.

\texttt{target_du_pause_element_token}

Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies the pause element to release the target task. Any PET that specifies a pause element not currently being used to pause a task is valid. When a PET for a previously released pause element is used to try to pause a task, the task is not paused; however, the value specified in target_du_release_code will still be returned to the caller of pause or transfer service.

If the task was paused and is now dispatchable, the task will immediately be given control on the current processor.

\textbf{CAUTION:}
Do not use the same PET for both current_du_pause_element_token and target_du_pause_element_token.

\texttt{target_du_release_code}

Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the caller of the pause or transfer service that used (or will use) the same PET to pause a task.

\textbf{ABEND codes}

None.

\textbf{Return codes}

When the service returns control to the resource manager, GPR 15 and the return_code parameter contain a hexadecimal return code.

<table>
<thead>
<tr>
<th>Return Code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00 (00)                       | IEA_SUCCESS   | \textbf{Meaning}: Successful completion.  
|                               |               | \textbf{Action}: None. |
### IEA4XFR callable service

<table>
<thead>
<tr>
<th>Return Code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04 (04)                      | IEA_PE_TOKEN_BAD      | **Meaning:** Program error. The specified pause element token is not valid. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 08 (08)                      | IEA_PE_TOKEN_STALE    | **Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 12 (0C)                      | IEA_DUPLICATE_PAUSE   | **Meaning:** The work unit has already been paused using the specified pause element token. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 16 (10)                      | IEA_SLEEP_DISRUPTED   | **Meaning:** RTM has terminated the task or SRB; no release is necessary.  
**Action:** None |
| 20 (14)                      | IEA_SPACE_TERMINATING | **Meaning:** The address space that contains the task or SRB is terminating; no release is necessary.  
**Action:** None |
| 24 (18)                      | IEA_LOCK_HELD         | **Meaning:** Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error. Correct the program and rerun it. |
| 32 (20)                      | IEA_PE_BAD_STATE      | **Meaning:** Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call.  
**Action:** Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it. |
| 36 (24)                      | IEA_UNSUPPORTED_MVS_RELEASE | **Meaning:** Environmental error. The system release does not support this service. The system rejects the service call.  
**Action:** Run the program on a system that supports the service. |
### IEA4XFR callable service

<table>
<thead>
<tr>
<th>Return Code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 (28)</td>
<td>IEA_INVALID_AUTHCODE</td>
<td><strong>Meaning:</strong> Program error. The auth_level value specified in the call is not valid. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>IEA_INVALID_MODE</td>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>52 (34)</td>
<td>IEA_ALREADY_SUSPENDED</td>
<td><strong>Meaning:</strong> The pause element was already paused. <strong>Action:</strong> Check the calling program for a probable coding error and correct the program and rerun it.</td>
</tr>
<tr>
<td>60 (3C)</td>
<td>IEA_AUTH_TOKEN</td>
<td><strong>Meaning:</strong> Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. <strong>Action:</strong> Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td>64 (40)</td>
<td>IEA_PE_NOT_HOME</td>
<td><strong>Meaning:</strong> Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>68 (44)</td>
<td>IEA_XFER_TO_SELF</td>
<td><strong>Meaning:</strong> Program error. The specified current_du_pause_element_token and target_du_pause_element_token are the same. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>72 (48)</td>
<td>IEA_XFER_FAILED</td>
<td><strong>Meaning:</strong> The transfer failed, and the current_du_pause_element_token is no longer useable. <strong>Action:</strong> Reissue the transfer request using the updated_du_pause_element_token. Deallocate the current_du_pause_element_token.</td>
</tr>
</tbody>
</table>
### IEA4XFR callable service

<table>
<thead>
<tr>
<th>Return Code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 4095 (FFF)                   | IEA_UNEXPECTED_ERROR | **Meaning:** This service routine encountered an unexpected error. The system rejects this service request.  
**Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
IEA4XFR callable service
Chapter 73. IEA4XFR2 — Transfer service

Description
Call IEA4XFR2 service to release a paused task or SRB, and, when possible, give the task or SRB immediate control. This service can also, optionally, pause the task or SRB under which the transfer request is made. If the caller does not request that its task or SRB be paused, the caller’s task or SRB remains dispatchable.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td></td>
</tr>
<tr>
<td>Environment factor</td>
<td>Requirement</td>
</tr>
<tr>
<td>Minimum authorization:</td>
<td></td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td></td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td></td>
</tr>
<tr>
<td>AMODE:</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td></td>
</tr>
<tr>
<td>Control parameters:</td>
<td></td>
</tr>
</tbody>
</table>

Programming requirements
Either link the calling program’s object code with the linkable stub routine (IEA4CSS from SYS1.CSSLIB) or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

<table>
<thead>
<tr>
<th>HLL Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEAASM</td>
<td>390 Assembler declarations</td>
</tr>
<tr>
<td>IEAC</td>
<td>C/390 and C++/390 declarations</td>
</tr>
</tbody>
</table>

Restrictions
Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.
IEA4XFR2 callable service

Transfer cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information
Before calling the Transfer service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter address list.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a 144-byte register save area.</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSSSTATE AMODE64=YES</td>
<td></td>
</tr>
<tr>
<td>CALL IEA4XFR2</td>
<td>,return_code, ,current_du_pause_element_token, ,updated_pause_element_token, ,current_du_release_code, ,target_du_pause_element_token, ,target_du_release_code, ,linkage)</td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

**return_code**
- Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Transfer service.

**current_du_pause_element_token**
- Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies the pause element that is being or will be used to pause a task or SRB. When a PET is used on a call to the pause service, it cannot be reused on a second call to pause or as a current_du_pause_element_token on transfer. A new PET is returned to update_pause_element_token. The new PET now properly defines the pause element and should be used the next time when a pause, transfer, release, or deallocate_pause_element request is using the same pause element.

If the value specified is 16-bytes of binary zeros, the current task or SRB will not be paused. The updated_pause_element_token and current_du_release_code are unpredictable.

CAUTION:
Do not specify the same PET for both current_du_pause_element_token and target_pause_element_token.

**updated_pause_element_token**
- Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a new pause element token that identifies the pause element originally identified by the PET specified in current_du_pause_element_token. The PET originally specified in current_du_pause_element_token cannot be reused after a successful call to Pause or Transfer.

If you set the current_du_pause_element_token to zeros, the contents of updated_pause_element_token are unpredictable.

**current_du_release_code**
- Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code set by the issuer of the release or transfer service that released the current task or SRB from the paused condition.
IEA4XFR2 callable service

If you set the current_du_pause_element_token to zero, the contents are unpredictable.

\texttt{target_du_pause_element_token}
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies a pause element that is being or will be used to pause a task or SRB. If the task or SRB is paused, it will be released, and, if possible, be given control. If the task or SRB is not paused using the specified pause element, it will not be paused when an attempt to pause is made. In either case the task or SRB will be returned the value specified in target_release_code.

\textbf{CAUTION:}
Do not use the same PET for both current_du_pause_element_token and target_du_pause_element_token.

\texttt{target_du_release_code}
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the caller of the pause or transfer service used (or will use) the PET specified in target_du_pause_element_token to pause a task or SRB.

\texttt{linkage}
Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Transfer service routine is to be invoked. The following options are supported:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value (hexadecimal)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA_LINKAGE_SVC</td>
<td>0</td>
<td>The Transfer service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and either problem state or supervisor state.</td>
</tr>
<tr>
<td>IEA_LINKAGE_BRANCH</td>
<td>1</td>
<td>The Transfer service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.</td>
</tr>
</tbody>
</table>

\textbf{ABEND codes}
None.

\textbf{Return codes}
When the service returns control to the resource manager, GPR 15 and the return_code parameter contain a hexadecimal return code.
### IEA4XFR2 callable service

<table>
<thead>
<tr>
<th>Return Code in:</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal (Hex)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 (00)</td>
<td>IEA_SUCCESS</td>
<td><strong>Meaning:</strong> Successful completion. <strong>Action:</strong> None</td>
</tr>
<tr>
<td>04 (04)</td>
<td>IEA_PE_TOKEN_BAD</td>
<td><strong>Meaning:</strong> Program error. The specified pause element token is not valid. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>IEA_PE_TOKEN_STALE</td>
<td><strong>Meaning:</strong> The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>12 (0C)</td>
<td>IEA_DUPLICATE_PAUSE</td>
<td><strong>Meaning:</strong> The work unit has already been paused using the specified pause element token. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>16 (10)</td>
<td>IEA_SLEEP_DISRUPTED</td>
<td><strong>Meaning:</strong> RTM has terminated the task or SRB; no release is necessary. <strong>Action:</strong> None</td>
</tr>
<tr>
<td>20 (14)</td>
<td>IEA_SPACE_TERMINATING</td>
<td><strong>Meaning:</strong> The address space that contains the task or SRB is terminating; no release is necessary. <strong>Action:</strong> None</td>
</tr>
<tr>
<td>24 (18)</td>
<td>IEA_LOCK HELD</td>
<td><strong>Meaning:</strong> Program error. If a current_du_pause_element_token of 16 bytes of binary zeros is specified, one or more locks other than the local lock are held. Otherwise, one or more locks are held. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>32 (20)</td>
<td>IEA_PE_BAD_STATE</td>
<td><strong>Meaning:</strong> Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it.</td>
</tr>
<tr>
<td>36 (24)</td>
<td>IEA_UNSUPPORTED_MVS_RELEASE</td>
<td><strong>Meaning:</strong> Environmental error. The system release does not support this service. The system rejects the service call. <strong>Action:</strong> Run the program on a system that supports the service.</td>
</tr>
<tr>
<td>44 (2C)</td>
<td>IEA_INVALID_MODE</td>
<td><strong>Meaning:</strong> Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. <strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
</tbody>
</table>
### IEA4XFR2 callable service

<table>
<thead>
<tr>
<th>Return Code in: Decimal (Hex)</th>
<th>Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>52 (34)</td>
<td>IEA_ALREADY_SUSPENDED</td>
<td><strong>Meaning:</strong> The pause element was already paused.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error and correct the program and rerun it.</td>
</tr>
<tr>
<td>60 (3C)</td>
<td>IEA_AUTH_TOKEN</td>
<td><strong>Meaning:</strong> Program error. The caller was in Problem state or key 8, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Program error. The specified pause element token is not valid. The system rejects the service call.</td>
</tr>
<tr>
<td>64 (40)</td>
<td>IEA_PE_NOT_HOME</td>
<td><strong>Meaning:</strong> Program error. The specified pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>68 (44)</td>
<td>IEA_XFER_TO_SELF</td>
<td><strong>Meaning:</strong> Program error. The specified current_du_pause_element_token and target_du_pause_element_token are the same.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the calling program for a probable coding error. Correct the program and rerun it.</td>
</tr>
<tr>
<td>72 (48)</td>
<td>IEA_XFER_FAILED</td>
<td><strong>Meaning:</strong> The transfer failed, and the current_du_pause_element_token is no longer usable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Reissue the transfer request using the updated_du_pause_element_token. Deallocate the current_du_pause_element_token.</td>
</tr>
<tr>
<td>4095 (FFF)</td>
<td>IEA_UNEXPECTED_ERROR</td>
<td><strong>Meaning:</strong> This service routine encountered an unexpected error. The system rejects this service request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</td>
</tr>
</tbody>
</table>
Chapter 74. IEECMDS — Query/remove attached commands

Description
This macro provides the same function as the CMDS operator command. It can be used to obtain information about MVS commands which are attached or waiting to be attached in the "MASTER" or CONSOLE address spaces.

It can also be used to remove commands which are waiting. It cannot be used to cancel commands which are already executing.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state. System PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN = SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit addressing mode.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No Locks may be held.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements
The calling program may include mapping macro IEEZB889 which can be used to map the information which is returned in the buffer specified as BUFFER.

Restrictions
The caller cannot be protected by an FRR.

Input register information
Before issuing the IEECMDS macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason Code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:
### IEECMDS macro

#### Register

**Contents**

- **0-1** Used as work registers by the system
- **2-13** Unchanged
- **14–15** Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

#### Performance implications

None.

#### Syntax

The IEECMDS macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: Symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede IEECMDS.</td>
</tr>
<tr>
<td><strong>IEECMDS</strong></td>
<td></td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must follow IEECMDS.</td>
</tr>
<tr>
<td><code>REQUEST=COUNT</code></td>
<td>Default: REQUEST=COUNT</td>
</tr>
<tr>
<td><code>,REQUEST=INFO</code></td>
<td></td>
</tr>
<tr>
<td><code>,REQUEST=REMOVE</code></td>
<td></td>
</tr>
<tr>
<td><code>,BUFFER=buffer</code></td>
<td>Required with REQUEST=INFO or REQUEST=REMOVE</td>
</tr>
<tr>
<td><code>buffer</code></td>
<td><code>buffer</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>,BUFSIZE=bufsize</code></td>
<td>Required with REQUEST=INFO or REQUEST=REMOVE</td>
</tr>
<tr>
<td><code>bufsize</code></td>
<td><code>bufsize</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>,CLASS=class</code></td>
<td><code>class</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>,CLASS=ANY_CLASS</code></td>
<td>Default: CLASS=ANY_CLASS</td>
</tr>
<tr>
<td><code>,CMD=cmd</code></td>
<td><code>cmd</code>: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td><code>,CMD=ANY_CMD</code></td>
<td>Default: CMD=ANY_CMD</td>
</tr>
<tr>
<td><code>,ID =id</code></td>
<td><code>id</code>: RS-type address or address in register (2) - (12)</td>
</tr>
</tbody>
</table>

---

762  z/OS V2R1.0 MVS Authorized Assembler Services Reference EDT-IXG
### Parameters

The parameters are explained as follows:

- **name**
  - An optional symbol, starting in column 1, that is the name on the IEECMDS macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

- **REQUEST=COUNT**
- **REQUEST=INFO**
- **REQUEST=REMOVE**
  - An optional parameter that indicates the type of request. The default is REQUEST=COUNT.
  - **REQUEST=COUNT**
    - Return only the count of commands which meet the search criteria.
    - This count is returned for all values of REQUEST, but if REQUEST=COUNT, no other information is returned.
  - **REQUEST=INFO**
IEECMDS macro

- Return information about commands meeting the search criteria.
- The count of matching commands is returned.
- The following information is returned for each command:
  - command names
  - id numbers
  - "waiting or executing" status
  - jobname and asid of the command issuer
  - date/time of issue/execution

- REQUEST=REMOVE
- Remove commands from the "waiting for execution" status, if they meet the search criteria.
- The count of matching commands is returned.
- The following information is returned for each removed command:
  - command names
  - id numbers
  - "waiting or executing" status
  - jobname and asid of the command issuer
  - date/time of issue/execution

Message IEE065I is issued for each removed command. It is directed to the console that issued the removed command.

BUFFER=buffer
A required input parameter if REQUEST=INFO or REQUEST=REMOVE is specified to contain the response.

To code: Specify the RS-type address of address in register (2)-(12), of a character field.

BUFSIZE=bufsize
A required input parameter if REQUEST=INFO or REQUEST=REMOVE is specified to contain the size of the output storage buffer.

To code: Specify the RS-type address of address in register (2)-(12), of a fullword field.

CLASS=class
CLASS=ANY_CLASS
An optional input parameter that indicates the class of commands to be processed.

The currently defined classes are:
- Class M1 — commands which are attached to *MASTER*, and may be essential to clearing a backlog of Class M2 commands.
- Class M2 — ordinary attached commands which run in the *MASTER* address space.
- Class M3 — only for SEND commands which run in the *MASTER* address space.
- Class C1 — commands which are attached in CONSOLE, and may be essential to clearing a backlog of Class C2 commands.
- Class C2 — ordinary attached commands which run in the CONSOLE address space.
- Class C3 — only for ROUTE commands which run in the CONSOLE address space.
IEECMD macro

The default is ANY_CLASS.

For detailed information about command classes, see the description of command flooding in z/OS MVS System Commands.

To code: Specify the RS-type address or address in register (2)-(12) of a 4–character field.

CMD=cmd
CMD=ANY_CMD

An optional input parameter that indicates the name of command to be processed.

The command name must be specified as the full name, not an abbreviation. This is to conform with the command name returned during the previous execution of the macro with REQUEST=INFO. The default is ANY_CMD.

To code: Specify the RS-type address or address in register (2)-(12) of an 8–character field.

ID=id
ID=ANY_ID

An optional input parameter that indicates the id number of the command that had been returned on a previous CMDS INFO command. The default is ANY_ID.

To code: Specify the RS-type address or address in register (2)-(12) of a 4–character field.

JOB=job
JOB=ANY_JOB

An optional input parameter that indicates the jobname of the job which issued the commands. The default is ANY_CMD.

To code: Specify the RS-type address or address in register (2)-(12) of an 8–character field.

COUNT=count

An required input parameter that contains the address area that will contain the number of commands meeting the specified criteria.

To code: Specify the RS-type address or address in register (2)-(12) of an pointer field.

,RETCODE=retcode

An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=rsncode

An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:
IEECMDS macro

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **1**, if you use the currently available parameters.

**To code:** Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 1

```
,(MF=S,
   (L,list addr)
   (L,list addr,attr)
   (L,list addr,0D)
   (E,list addr)
   (E,list addr,COMPLETE)
   (E,list addr,NOCHECK)
   (M,list addr)
   (M,list addr,COMPLETE)
   (M,list addr,NOCHECK)
```

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

```
,list addr
```

The name of a storage area to contain the parameters. For MF=S MF=E and MF=M, this can be an RS-type address or an address in register (1)-(12).
IEECMDS macro

,.attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code .attr, the system provides a value of 0D.

,.COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,.NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

IBM recommends that you use the modify and execute forms of IEECMDS in the following order:
1. Use IEECMDS...MF=(M,list-addr,COMPLETE) specifying appropriate parameters, including all required ones.
2. Use IEECMDS...MF=(M,list-addr,NOCHECK) specifying the parameters that you want to change.
3. Use IEECMDS...MF=(E,list-addr,NOCHECK) to execute the macro.

ABEND codes
None.

Return codes
Macro IEECMDS provides equate symbols for the return and reason codes.

When the IEECMDS macro returns control to your program:
• GPR 15 (and retcode, when you code RETCODE) contains a return code.
• When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>—</td>
<td>Equate Symbol: CMDS_RC_OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: Matching commands have been found. In the case of a REQUEST type of INFO or REMOVE, the output buffer was sufficient to hold all of the information for the commands meeting the search criteria.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None</td>
</tr>
<tr>
<td>04</td>
<td>—</td>
<td>Equate Symbol: CMDS_RC_NOCMDS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: No commands meet the specified filters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None</td>
</tr>
<tr>
<td>08</td>
<td>—</td>
<td>Equate Symbol: CMDS_RC_NOSTOR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: Insufficient return buffer storage to complete the query operation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Refer to the action provided with the specific reason code.</td>
</tr>
</tbody>
</table>
### IEECMDS macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
</table>
| 08                      | 04                      | **Equate Symbol**: CMDS_RS_SOMECMDS  
**Meaning**: The output buffer is too small to contain all requested information, but does contain the information for one or more commands.  
If REQUEST=REMOVE, the system has removed only the commands for which information is returned.  
**Action**: The count of matching commands has been returned. Adjust the buffer size so that it is at least as large as the count multiplied by the output size for each entry, plus the length of the header, and issue the macro again.  
The constant CMDS_HEADER_LENGTH represents the amount of storage required for the buffer header.  
The constant CMDS_ENTRY_LENGTH represents the amount of storage required per command.  
These constants are declared in mapping macro IEEZB889. |
| 08                      | 08                      | **Equate Symbol**: CMDS_RS_NOCMDS  
**Meaning**: The output buffer is too small to contain the information for even one command.  
If REQUEST=REMOVE, the system has not removed any commands.  
**Action**: The count of matching commands has been returned. Adjust the buffer size so that it is at least as large as the count multiplied by the output size for each entry, plus the length of the header, and issue the macro again.  
The constant CMDS_HEADER_LENGTH represents the amount of storage required for the buffer header.  
The constant CMDS_ENTRY_LENGTH represents the amount of storage required per command.  
These constants are declared in mapping macro IEEZB889. |
| 10                      | —                       | **Equate Symbol**: CMDS_RC_INVPL  
**Meaning**: Invalid parameter list.  
**Action**: Refer to the action provided with the specific reason code. |
| 10                      | 04                      | **Equate Symbol**: CMDS_RS_INVACRN  
**Meaning**: The acronym in the parameter list was invalid.  
**Action**: Correct the acronym in the parameter list and issue IEECMDS again. |
| 10                      | 08                      | **Equate Symbol**: CMDS_RS_INVADDR  
**Meaning**: An output message is invalid. An ABEND occurred while trying to access storage at an address specified in the parameter list, possibly because that storage is not accessible by the caller, or the storage does not exist.  
**Action**: Correct the invalid address in the parameter list and issue IEECMDS again. |
### IEECMDS macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0C</td>
<td><strong>CMDS_RS_INVBUFFER</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The address or length of the buffer in the parameter list was invalid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Correct the values of BUFFER or BUFSIZE or both in the parameter list and issue IEECMDS again.</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td><strong>CMDS_RS_INVLNGTH</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The length of the parameter list is invalid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Correct the length in the parameter list and issue IEECMDS again.</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td><strong>CMDS_RS_INVVERS</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The version specified in the parameter list is invalid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Correct the version in the parameter list and issue IEECMDS again.</td>
</tr>
<tr>
<td>10</td>
<td>18</td>
<td><strong>CMDS_RS_INVFUNC</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The REQUEST type specified in the parameter list is not a valid REQUEST type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Correct the REQUEST type in the parameter list and issue IEECMDS again.</td>
</tr>
<tr>
<td>10</td>
<td>1C</td>
<td><strong>CMDS_RS_INVCLASS</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The CLASS specified in the parameter list is not a valid CLASS name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Correct the CLASS in the parameter list and issue IEECMDS again.</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td><strong>CMDS_RS_INVID</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The ID specified in the parameter list is not a valid value. The ID value must be a decimal number in EBCDIC printable (EBCDIC) characters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Correct the ID in the parameter list and issue IEECMDS again.</td>
</tr>
<tr>
<td>40</td>
<td>—</td>
<td><strong>CMDS_RS_SYSERR</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> System Error. This return code is for IBM diagnostic purposes only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Record the return and reason codes and supply it to the appropriate IBM support personnel.</td>
</tr>
<tr>
<td>40</td>
<td>04</td>
<td><strong>CMDS_RS_SYSABEND</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> An ABEND occurred during processing. This reason code is for IBM diagnostic purposes only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Record the return and reason codes and supply it to the appropriate IBM support personnel.</td>
</tr>
<tr>
<td>40</td>
<td>08</td>
<td><strong>CMDS_RS_SYSERR</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> An error occurred during processing. This reason code is for IBM diagnostic purposes only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Record the return and reason codes and supply it to the appropriate IBM support personnel.</td>
</tr>
</tbody>
</table>
IEECMDS macro
Chapter 75. IEEQEMCS — Query EMCS console

Description

This macro returns information about EMCS consoles in the system.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state. System PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks may be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

The calling program may include mapping macro IEEZB887 which can be used to map the information that is returned in the buffer addressed by BUFPTR.

The calling program may also optionally include macro IEEZB888, which contains the declarations for the reason and return codes used by IEEQEMCS.

Restrictions

None.

Input register information

Before issuing the IEEQEMCS macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:
IEEQEMCS macro

Register

Contents

0-1   Used as work registers by the system
2-13  Unchanged
14-15 Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

If REQUEST=FULL is specified, the service must read the data space of each console being reported on. Depending on the number of consoles, this can degrade performance of the service.

Syntax

The IEEQEMCS macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEEQEMCS.</td>
</tr>
<tr>
<td>IEEQEMCS</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEEQEMCS.</td>
</tr>
<tr>
<td>REQUEST=COUNT</td>
<td>Default: REQUEST=COUNT</td>
</tr>
<tr>
<td>REQUEST=SUMMARY</td>
<td></td>
</tr>
<tr>
<td>,BUFPRTR=bufptr</td>
<td>bufptr: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,BUFSIZE=bufsize</td>
<td>bufsize: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,TOKEN=token</td>
<td>token: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,RECSIZE=recesize</td>
<td>recsize: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>REQUEST=INFO</td>
<td></td>
</tr>
<tr>
<td>,BUFPRTR=bufptr</td>
<td>bufptr: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,BUFSIZE=bufsize</td>
<td>bufsize: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,TOKEN=token</td>
<td>token: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,RECSIZE=recesize</td>
<td>recsize: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>REQUEST=FULL</td>
<td></td>
</tr>
<tr>
<td>,BUFPRTR=bufptr</td>
<td>bufptr: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,BUFSIZE=bufsize</td>
<td>bufsize: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,TOKEN=token</td>
<td>token: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,RECSIZE=recesize</td>
<td>recsize: RS-type address or address in register (2) - (12).</td>
</tr>
</tbody>
</table>
### IEEQEMCS macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_STATUS=ACTIVE</td>
<td>Default: STATUS=ACTIVE</td>
</tr>
<tr>
<td>_STATUS=INACTIVE</td>
<td></td>
</tr>
<tr>
<td>_STATUS=ALL</td>
<td></td>
</tr>
<tr>
<td>_STATUS=BACKLOG</td>
<td></td>
</tr>
<tr>
<td>_STATUS=ERR</td>
<td></td>
</tr>
<tr>
<td>_BKLG_NUM=blk_num</td>
<td><code>blk_num</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td>Default: BKLG_NUM=10</td>
</tr>
<tr>
<td>_CN=cn</td>
<td><code>cn</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_SYS=sys</td>
<td><code>sys</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_KEY=key</td>
<td><code>key</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_AUTH=ANY</td>
<td>Default: AUTH=ANY</td>
</tr>
<tr>
<td>_AUTH=MASTER</td>
<td></td>
</tr>
<tr>
<td>_AUTH=SYS</td>
<td></td>
</tr>
<tr>
<td>_AUTH=IO</td>
<td></td>
</tr>
<tr>
<td>_AUTH=CONS</td>
<td></td>
</tr>
<tr>
<td>_AUTH=ALL</td>
<td></td>
</tr>
<tr>
<td>_AUTH=INFO</td>
<td></td>
</tr>
<tr>
<td>_AUTH=SYSONLY</td>
<td></td>
</tr>
<tr>
<td>_AUTH=IOONLY</td>
<td></td>
</tr>
<tr>
<td>_AUTH=CONSONLY</td>
<td></td>
</tr>
<tr>
<td>_AUTH=ALLONLY</td>
<td></td>
</tr>
<tr>
<td>_AUTH=INFOONLY</td>
<td></td>
</tr>
<tr>
<td>_ATTR=ANY</td>
<td>Default: ATTR=ANY</td>
</tr>
<tr>
<td>_ATTR=YES</td>
<td></td>
</tr>
<tr>
<td>_ATTR=ROUT</td>
<td></td>
</tr>
<tr>
<td>_ATTR=HC</td>
<td></td>
</tr>
<tr>
<td>_ATTR=AUTO</td>
<td></td>
</tr>
<tr>
<td>_ATTR=AUTON</td>
<td></td>
</tr>
<tr>
<td>_ATTR=MN</td>
<td></td>
</tr>
<tr>
<td>_ATTR=INTIDS</td>
<td></td>
</tr>
<tr>
<td>_ATTR=UNKNIDS</td>
<td></td>
</tr>
<tr>
<td>_ATTR=NONE</td>
<td></td>
</tr>
<tr>
<td>_DOM=ANY</td>
<td>Default: DOM=ANY</td>
</tr>
</tbody>
</table>

Chapter 75. IEEQEMCS — Query EMCS console  773
### IEEQEMCS macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.DOM=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>.DOM=NORMAL</code></td>
<td></td>
</tr>
<tr>
<td><code>.DOM=ALL</code></td>
<td></td>
</tr>
<tr>
<td><code>.DOM=NONE</code></td>
<td></td>
</tr>
<tr>
<td><code>.CONSCNT=consent</code></td>
<td><code>consent</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>.RETCODE=retcode</code></td>
<td><code>retcode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>.RSNCODE=rsncode</code></td>
<td><code>rsncode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>.PLISTVER=IMPLIED_VERSION</code></td>
<td>Default: <code>PLISTVER=IMPLIED_VERSION</code></td>
</tr>
<tr>
<td><code>.PLISTVER=MAX</code></td>
<td></td>
</tr>
<tr>
<td><code>.PLISTVER=1</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(L,list addr)</code></td>
<td><code>list addr</code>: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td><code>.MF=(L,list addr,attr)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(L,list addr,0D)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(E,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(E,list addr,COMPLETE)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(E,list addr,NOCHECK)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(M,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(M,list addr,COMPLETE)</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(M,list addr,NOCHECK)</code></td>
<td></td>
</tr>
</tbody>
</table>

#### Parameters

The parameters are explained as follows:

**name**

An optional symbol, starting in column 1, that is the name on the IEEQEMCS macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=COUNT
REQUEST=SUMMARY
REQUEST=INFO
REQUEST=FULL

An optional parameter that indicates the type of information request. The default is REQUEST=COUNT.
**REQUEST=COUNT**
Return only the number of EMCS consoles meeting the search criteria.

**REQUEST=SUMMARY**
Return the number and names of the consoles that meet the search criteria.

,BUFPTR=bufptr
An optional input parameter that contains the address of the storage that the console display will be returned in. This field is only valid for a SUMMARY, INFO, or FULL request.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,BUFSIZE=buFSIZE
When BUFPTR=bufptr is specified, a required input parameter that contains the size of the storage buffer. This field is only valid when BUFPTR is specified.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,TOKEN=token
An optional input parameter that returns the address of an 8-byte token used to return additional EMCS console information on subsequent calls if BUFSIZE is insufficient. This field is only valid for a SUMMARY, INFO, or FULL request.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,RECSIZE=recsize
When TOKEN is specified, RECSIZE is a required input parameter that contains the address of a 4-byte output area that will contain the recommended size of the output storage buffer if BUFSIZE is insufficient. This size represents only enough storage to store information about the one console represented by the TOKEN at the time of this call to IEEQEMCS. The RECSIZE contains data only when return code = is IEEQE_RC_NOSTOR.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

**REQUEST=INFO**
Return the number, names, and console data information, but no message data space statistics for the consoles meeting the search criteria.

**REQUEST=FULL**
Return the number, names, console data information, and message data space statistics for the consoles meeting the search criteria.

,STATUS=ACTIVE
,STATUS=INACTIVE
,STATUS=ALL
,STATUS=BACKLOG
,STATUS=ERR
An optional parameter that indicates the status of the EMCS consoles to be returned. The default is STATUS=ACTIVE.

,STATUS=ACTIVE
Return only active consoles.
IEEQEMCS macro

,STATUS=INACTIVE
   Return only inactive consoles.

,STATUS=ALL
   Return both active and inactive consoles.

,STATUS=BACKLOG
   Return consoles with unretrieved messages. The BKLG_NUM keyword can
   specify the minimum number of unretrieved messages a console must have
   to be returned.

,STATUS=ERR
   Return only EMCS consoles in an error condition.

,BKLG_NUM=bklg_num
   An optional input parameter that indicates the minimum number of
   unretrieved messages a console must have for it to be returned on a
   STATUS(BACKLOG) call.

   To code: Specify the RS-type address, or address in register (2)-(12), of a
   fullword field.

,CN=cn
   An optional input parameter that indicates a console name to search for. The
   name include wildcard characters.

   To code: Specify the RS-type address, or address in register (2)-(12), of an
   8-character field.

,SYs=sys
   An optional input parameter that indicates the system name where the console
   was last activated. The system name may include wildcarded characters.

   To code: Specify the RS-type address, or address in register (2)-(12), of an
   8-character field.

,KEY=key
   An optional input parameter that indicates the KEY that was used to activate
   the console. The key name may include wildcard characters.

   To code: Specify the RS-type address, or address in register (2)-(12), of an
   8-character field.

,AUTH=ANY
,AUTH=MASTER
,AUTH=SYS
,AUTH=IO
,AUTH=CONS
,AUTH=ALL
,AUTH=INFO
,AUTH=SYSONLY
,AUTH=TOONLY
,AUTH=CONSONLY
,AUTH=ALLONLY
,AUTH=INFOONLY
   An optional parameter that indicates console command authority. The default
   is AUTH=ANY.

   To code: Specify the RS-type address, or address in register (2)-(12), of an
   8-character field.

   AUTH=ANY
      Return consoles with any authority.

   AUTH=MASTER
      Return consoles with MASTER authority only.
IEEQEMCS macro

\texttt{AUTH=SYS}
Return consoles with SYS authority or MASTER authority.

\texttt{AUTH=IO}
Return consoles with IO authority or MASTER authority.

\texttt{AUTH=CONS}
Return consoles with CONS authority or MASTER authority.

\texttt{AUTH=ALL}
Return consoles with IO, SYS, and CONS authority, or MASTER authority.

\texttt{AUTH=INFO}
Return consoles with INFO, IO, SYS, CONS, or MASTER authority.

\texttt{AUTH=SYSONLY}
Return consoles with SYS authority only.

\texttt{AUTH=IOONLY}
Return consoles with IO authority only.

\texttt{AUTH=CONSONLY}
Return consoles with CONS authority only.

\texttt{AUTH=ALLONLY}
Return consoles with IO, SYS, and CONS authority only.

\texttt{AUTH=INFOONLY}
Return consoles with INFO authority only.

\texttt{ATTR=ANY}
\texttt{ATTR=YES}
\texttt{ATTR=ROUT}
\texttt{ATTR=HC}
\texttt{ATTR=AUTO}
\texttt{ATTR=AUTON}
\texttt{ATTR=MN}
\texttt{ATTR=NONE}
\texttt{ATTR=INTIDS}
\texttt{ATTR=UNKNIDS}
An optional parameter that indicates routing attributes of the console. The default is \texttt{ATTR=ANY}.

\texttt{ATTR=ANY}
Return consoles regardless of routing attributes.

\texttt{ATTR=YES}
Return consoles that are receiving some type of undelivered messages.

\texttt{ATTR=ROUT}
Return consoles receiving any routing codes.

\texttt{ATTR=HC}
Return consoles receiving the hardcopy message set.

\texttt{ATTR=AUTO}
Return consoles receiving AUTO(YES) messages.

\texttt{ATTR=AUTON}
Return consoles not receiving AUTO(YES) messages.

\texttt{ATTR=MN}
Return consoles receiving any type of MONITOR messages.
IEEQEMCS macro

,ATTR=None
  Return consoles with no routing attributes.

,ATTR=INTIDS
  Return consoles receiving messages directed to console id zero.

,ATTR=UNKNIDS
  Return consoles receiving messages directed to "unknown" console ids,
  such as consoles with one-byte id.

,DOM=ANY
,DOM=NO
,DOM=YES
,DOM=NORMAL
,DOM=ALL
,DOM=NONE
  An optional parameter that indicates the DOM attribute of the consoles. The
default is DOM=ANY.

,DOM=ANY
  Return consoles regardless of their DOM attribute.

,DOM=YES
  Return consoles that are receiving DOMs (either DOM(NORMAL) or
  DOM(NONE) consoles).

,DOM=NORMAL
  Return consoles that are DOM(NORMAL) only.

,DOM=ALL
  Return consoles that are DOM(ALL) only.

,DOM=NONE
  Return consoles that are DOM(NONE) only.

,CONSCNT=consnt
  An optional input parameter that contains the address of a 4-byte output area
  that will contain the number of consoles meeting the specified criteria. The
  output area is only filled in for a COUNT request.

  To code: Specify the RS-type address, or address in register (2)-(12), of a
  pointer field.

,RETCODE=retcode
  An optional parameter into which the return code is to be copied from GPR 15.

  To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=rsncode
  An optional output parameter into which the reason code is to be copied from
  GPR 0.

  To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
  An optional input parameter that specifies the version of the macro. PLISTVER
determines which parameter list the system generates. PLISTVER is an
optional input parameter on all forms of the macro, including the list form.
When using PLISTVER, specify it on all macro forms used for a request and
with the same value on all of the macro forms. The values are:
IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

1 The currently available parameters.

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 1

,MF=(L,list addr)
,MF=(L,list addr,attr)
,MF=(L,list addr,0D)
,MF=(E,list addr)
,MF=(E,list addr,COMPLETE)
,MF=(E,list addr,NOCHECK)
,MF=(M,list addr)
,MF=(M,list addr,COMPLETE)
,MF=(M,list addr,NOCHECK)

A required input parameter that specifies the macro form.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms of IEEQEMCS in the following order:
- Use IEEQEMCS ...MF=(M,list-addr,COMPLETE) specifying appropriate parameters, including all required ones.
- Use IEEQEMCS ...MF=(M,list-addr,NOCHECK), specifying the parameters that you want to change.
IEEQEMCS macro

- Use IEEQEMCS ...MF=(E,list-addr,NOCHECK), to execute the macro.

-,list addr
The name of a storage area to contain the parameters. For MF=E and MF=M, this can be an RS-type address or an address in register (1)-(12).

-,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

-,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

-,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

ABEND codes
None.

Return and reason codes
When the IEEQEMCS macro returns control to your program:
- GPR 15 (and retcode, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>—</td>
<td>Equate Symbol: IEEQE_RC_OK</td>
<td>Meaning: EMCS consoles have been found. In the case of a REQUEST type of SUMMARY, INFO, or FULL, the output buffer was large enough sufficient to hold all of the information for the consoles meeting the search criteria. Action: None required.</td>
</tr>
<tr>
<td>4</td>
<td>—</td>
<td>Equate Symbol: IEEQE_RC_NOCONS</td>
<td>Meaning: No EMCS consoles meet the specified filters. Action: None required.</td>
</tr>
<tr>
<td>8</td>
<td>—</td>
<td>Equate Symbol: IEEQE_RC_NOSTOR</td>
<td>Meaning: Insufficient return buffer storage to complete the query operation. Action: Refer to the action provided with the specific reason code.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Reason Code</td>
<td>Equate Symbol</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 8           | 4           | IEEQE_RS_TOKSZCONS     | **Meaning**: A token and recommended buffer size have been returned in TOKEN and RECSIZE. Also, some console information has been returned in the output buffer.  
**Action**: Process the information returned in the console buffer, then issue IEEQEMCS again with the token that was returned by this call to IEEQEMCS to obtain more console information. |
| 8           | 8           | IEEQE_RS_TOKSZNOCONS   | **Meaning**: A token and recommended buffer size have been returned in TOKEN and RECSIZE. The output buffer is too small to return any EMCS console information.  
**Action**: Allocate a new buffer that is at least the size returned in RECSIZE, and issue IEEQEMCS again with the new buffer and the token returned on the previous IEEQEMCS call. The recommended buffer size returned in RECSIZE is sufficient to hold only one console. It may be necessary to obtain a buffer larger than that to hold all of the consoles returned by IEEQEMCS. |
| 8           | 12          | IEEQE_RS_NOTOKSZRET    | **Meaning**: TOKEN and RECSIZE parameters were not coded on the macro invocation, so IEEQEMCS could not return a recommended buffer size to the caller. The buffer size specified by BUFSIZE was not sufficient to hold all of the consoles returned by IEEQEMCS.  
**Action**: Issue IEEQEMCS again with the TOKEN and RECSIZE parameters. |
| 12          | —           | IEEQE_RC_INVTOK        | **Meaning**: Invalid token in parameter list.  
**Action**: Issue IEEQEMCS again with a correct token or a token of zeros. |
| 16          | —           | IEEQE_RC_INVPL         | **Meaning**: Invalid parameter list.  
**Action**: Refer to the action provided with the specific reason code. |
| 16          | 4           | IEEQE_RS_INVACRN       | **Meaning**: The eyecatcher (ECDM) in the parameter list was invalid.  
**Action**: Correct the eyecatcher (ECDM) in the parameter list and issue IEEQEMCS again. |
| 16          | 8           | IEEQE_RS_INVADDR       | **Meaning**: An output address is invalid. An ABEND occurred while trying to access storage at an address specified in the parameter list, possibly because that storage is not accessible by the caller, or the storage does not exist.  
**Action**: Correct the address in the parameter list and issue IEEQEMCS again. |
### Table 69. Return and Reason Codes for the IEEQEMCS Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>12</td>
<td>IEEQE_RS_INVBUFSIZEADDR</td>
<td>The BUFSIZE parameter was invalid. Correct the BUFSIZE parameter and issue IEEQEMCS again.</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>IEEQE_RS_INVLGTH</td>
<td>The length of the parameter list is invalid. Correct the length in the parameter list and issue IEEQEMCS again.</td>
</tr>
<tr>
<td>16</td>
<td>20</td>
<td>IEEQE_RS_INVVERS</td>
<td>The version specified in PLISTVER is invalid. Correct the version and issue IEEQEMCS again.</td>
</tr>
<tr>
<td>16</td>
<td>24</td>
<td>IEEQE_RS_INVFUNC</td>
<td>The REQUEST type specified in the parameter list is not a valid REQUEST type. Correct the REQUEST type in the parameter list and issue IEEQEMCS again.</td>
</tr>
<tr>
<td>16</td>
<td>28</td>
<td>IEEQE_RS_INVSTAT</td>
<td>The STATUS type specified in the parameter list is not a valid STATUS type. Correct the STATUS type in the parameter list and issue IEEQEMCS again.</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>IEEQE_RS_INVAUTH</td>
<td>The command authority specified in the parameter list is not a valid command authority type. Correct the AUTH value in the parameter list and issue IEEQEMCS again.</td>
</tr>
<tr>
<td>16</td>
<td>36</td>
<td>IEEQE_RS_INVDOM</td>
<td>The DOM attribute specified in the parameter list is not a valid DOM attribute type. Correct the DOM attribute in the parameter list and issue IEEQEMCS again.</td>
</tr>
<tr>
<td>16</td>
<td>40</td>
<td>IEEQE_RS_INCONSIST</td>
<td>A set of parameters specified in the parameter list conflict with each other. Correct the parameter list to avoid conflicting parameters and issue IEEQEMCS again.</td>
</tr>
<tr>
<td>16</td>
<td>44</td>
<td>IEEQE_RS_INVATTR</td>
<td>The routing attributes specified in the parameter list are not valid routing attribute types. Correct the ATTR field in the parameter list and issue IEEQEMCS again.</td>
</tr>
</tbody>
</table>
### Table 69. Return and Reason Codes for the IEEQEMCS Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 64          | —           | Equate Symbol: IEEQE_RC_SYSERR | **Meaning**: System Error. This return code is for IBM diagnostic purposes only.  
**Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel. |
| 64          | 04          | Equate Symbol: IEEQE_RS_SYSABEND | **Meaning**: An ABEND occurred during processing. This reason code is for IBM diagnostic purposes only.  
**Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel. |
| 64          | 08          | Equate Symbol: IEEQE_RS_SYSERR | **Meaning**: An error occurred during processing. This reason code is for IBM diagnostic purposes only.  
**Action**: Record the return and reason codes and supply it to the appropriate IBM support personnel. |

### Examples

**Example**

Operation: This example requests FULL information about all consoles on system SYS01 that are in an error condition.

```
* Initialize BUFSIZE and TOKEN
L   REG9,INITSIZE
ST  REG9,_BUFSIZE
MVC  TOKEN,INITTOKEN
Set up addresses for IEEQEMCS
LA  REG9,TOKEN
ST  REG9,TOKENPTR
LA  REG9,RECSZ
ST  REG9,RECSZPTR
* Get storage for output buffer
STORAGE OBTAIN,LENGTH=BUFSIZE,ADDR=BUFPTR,COND=NO,LOC=ANY
* Issue IEEQEMCS
DOQEMCS  EQU *
  IEEQEMCS REQUEST=FULL, Full info X
  STATUS=ALL, Active or inactive consoles X
  TOKEN=TOKENPTR, Token X
  BUFPTR=BUFPTR, Buffer address X
  BUFSIZE=BUFSIZE, Buffer size X
  RECSIZE=RECSZPTR, Recommended size X
  RETCODE=RETCODE, Return code X
  RSNCODE=RSNCODE, Reason code X
  MF=(E,PLIST,COMPLETE)
* Check return and reason codes from IEEQEMCS
LA  REG14,IEEQE_RC_OK Check if all consoles have been returned
C  REG14,RETCODE
BE  ALLOK All consoles have been returned, so process them
LA  REG14,IEEQE_RC_NOCONS Check if no consoles have been returned
C  REG14,RETCODE
BE  DONE No consoles met the filter, so exit
```
IEEQEMCS macro

LA REG14,IEEQE_RC_NOSTOR  ; Check if the storage buffer could
C REG14,RETCODE          ; not hold all the consoles
BNE ERROR                ; No, there was some kind of error
LA REG14,IEEQE_RS_TOKSZCONS
C REG14,RSNCODE
BE SOMEOK                ; The buffer was too small to hold X
                         ; all the consoles meeting this X
                         ; filter, but IEEQEMCS put as many X
                         ; consoles as possible in the X
                         ; buffer. Process the consoles, X
                         ; and call IEEQEMCS again.
LA REG14,IEEQE_RS_TOKSZNOCONS
C REG14,RSNCODE
BE GETMORE               ; The buffer was too small for X
                         ; even one console. Free the X
                         ; buffer, get more storage, and X
                         ; call IEEQEMCS again.

ALLOK EQU *              ; Call PROCCONS to process the output buffer.
* Call PROCCONS to process the consoles returned in
* the output buffer.
LA REG15,PROCCONS
BALR REG14,REG15
* There are no more consoles to process, so exit
XR REG15,REG15           ; Zero return code
B DONE

SOMEOK EQU *             ; Call PROCCONS to process the consoles returned in
* the output buffer.
LA REG15,PROCCONS
BALR REG14,REG15
* There is more console information to be retrieved, so
* loop back to DOQEMCS.
B DOQEMCS

PROCCONS EQU *          ; Process consoles, and return to caller.
* Put size of summ block in R10
L REG8,ECDM_SUMM_INFO
CH REG8,0(REG8)          ; Is this an info block?
BE INFOBLK
L HDRREG,BUFPTR          ; Load pointer to buffer in R2
USING ECDM_HDR,HDRREG    ; This should be the ECDM header X
                         ; pointer
L SUMMREG,ECDM_HDR_SIZE
AR SUMMREG,HDRREG        ; Load the size of the header
                         ; Find the address of the summary X
                         ; block
LH REG15,ECDM_NENT
ST REG15,NUMCONS
USING ECDM_SUMM,SUMMREG
USING ECDM_INFO,INFOREG
USING ECDM_DSP,DSPREG
USING ECDM_CNSW,CNSWREG
USING ECDM_MSCP,MSCPREG

CONSLLOP LTR REG15,REG15   ; Is the number of entries zero?
BZ CONSDONE
L REG8,ECDM_SUMM_SIZE    ; Put size of summ block in R10
L ENDREG,ECDM_SUMM_CONS_SIZE
AR REG8,ENDREG           ; Get address of end of the X
                         ; this console in R11
AR ENDREG,SUMMREG        ; Get address of end of the X
                         ; console block in R11
BLKLOOK LR REG9,REG9     ; Copy work addr into R9
SR REG9,ENDREG           ; Are we at the end of the block?
BZ DOCONS                ; Yes, do the actual console X
                         ; processing
LA REG9,ECDM_TYPE_INFO   ; Load the type of block in R9
CH REG9,0(REG8)          ; Is this an info block?
BE INFOBLK               ; Yes, branch to info block
Chapter 75. IEEQEMCS — Query EMCS console

IEEQEMCS macro

LA  REG9,ECDM_TYPE_DSP      Load the type of block in R9
CH  REG9,0(REG8)           Is this a DSP block?
BE  DSPBLK                Yes, branch to data space block
LA  REG9,ECDM_TYPE_MSCP    Load the type of block in R9
CH  REG9,0(REG8)           Is this an MSCOPE block?
BE  MSCPBLK               Yes, branch to MSCOPE block
B   ERROR                 There was a bad type, so exit
INFOBLK LR  INFOREG,REG8   Load INFO block addr into R4
A   REG8,ECDM_INFO_SIZE   Increment size of block
B   BLKLOOP               Find next block
DSPBLK LR  DSPREG,REG8    Load DSP block addr into R5
A   REG8,ECDM_DSP_SIZE    Increment size of block
B   BLKLOOP               Find next block
MSCPBLK LR  MSCPREG,REG8  Load MSCP block addr into R6
A   REG8,ECDM_MSCP_SIZE   Increment size of block
B   BLKLOOP               Find next block
DOCONS EQU *              Console processing
*   At this point, do any processing on this individual
*   console that is necessary.
*   *
*   Now, decrement the console count, and process the
*   next console in the buffer (if there is one).
L   REG15,NUMCONS
BCTR REG15,0               Decrement console count
ST   REG15,NUMCONS
LR   SUMMREG,REG8          Get address of next block in R3
B   CONSLOOP              process next block
CONSDONE L   REG14,RETCADDR
BR   REG14
*
GETMORE EQU *             There was not enough storage to complete the request, so
*   get some more.
*   First delete the old storage
STORAGE RELEASE,LENGTH=BUFSIZE,ADDR=BUF PTR
*   Now, since IEEQEMCS returned a recommended storage
*   size, move that amount into BUFSIZE, and obtain the
*   storage buffer.
MVC   BUFSIZE,RECSZ
STORAGE OBTAIN,LENGTH=BUFSIZE,ADDR=BUFPTR,COND=NO,LOC=ANY
*   Loop to issue IEEQEMCS again
B   DOQEMCS               *
*
ERROR EQU *              Do any error handling here
*
*
DONE EQU *               IEEQEMCS found no more consoles, so release the output
*   buffer.
STORAGE RELEASE,LENGTH=BUFSIZE,ADDR=BUFPTR
*   Now, continue on with other processing, etc.
*
*
*   Declare constants and variables
BUFSIZE  DS  F
BUF PTR  DS  F
TOKENPTR DS  F
RECSZPTR DS  F
IEEQEMCS macro

TOKEN DS CLB
RECSZ DS F
RETCODE DS F
RSCODE DS F
NUMCONS DS F
READDR DS F
INITSIZE DC XL4'300'
INITTOKEN DC XL8'0'

* IEEQEMCS parameter list
  DS 0F
  IEEQEMCS MF=(L,PLIST)
* IEEQEMCS return code equates
  IEEZ8888
* Include mapping of ECDM blocks
  IEEZ8887
END
Chapter 76. IEEVARYD — Vary one or more devices online or offline

Description

IEEVARYD varies one or more devices online or offline on a single system, or defines the automatically switchable attribute for a device that supports automatic tape switching. It has the same effect as the VARY device or VARY AUTOSWITCH operator command, but it provides return and reason codes to the calling program, rather than issuing messages to a console.

See z/OS HCD Planning for more information about automatic tape switching, and z/OS Planning for Installation for the devices that support automatic tape switching.

Comparison to MGCRE macro

The MGCRE macro also allows a program to issue the VARY command. (MGCRE allows a program to issue any command.) However, MGCRE automatically checks the SAF authority of the calling program. Also, MGCRE returns text responses to a console specified by the calling program, whereas IEEVARYD provides a return and reason code.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PKM keys 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

- The caller must be in a non-swappable address space.
- The IEEVARYD macro does no authorization checking (SAF is not invoked). If you require that the calling program's authority to perform a vary device operation be checked, issue RACROUTE REQUEST=AUTH before issuing the IEEVARYD macro.
- The calling program must perform any logging of information associated with the operation, such as an entry in the system log (SYSLOG).
- You must include the IEEZB833 mapping macro and, if you specify the optional RESULTS keyword, the IEEZB834 mapping macro.
- You can change the automatically switchable characteristic of a tape device only if the device is offline.
IEEVAR YD macro

- The VDEV_ENQS_HELD flag that allowed the calling program to hold the SYSIEFSD.VARYDEV and SYSIEFSD.Q4 resources is no longer supported. The ENQs must be released before invoking IEEVAR YD or an abend 077-003C will result.

Restrictions

- The VDEV_ENQS_HELD flag that allowed the calling program to hold the SYSIEFSD.VARYDEV and SYSIEFSD.Q4 resources is no longer supported.
- The ENQs must be released before invoking IEEVAR YD or an abend 077-003C will result.

Input register information

Before issuing the IEEVAR YD macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IEEVAR YD macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
</tbody>
</table>
IEEVARYD macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEEVARYD.</td>
</tr>
<tr>
<td>IEEVARYD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEEVARYD.</td>
</tr>
<tr>
<td>operation parm</td>
<td>operation parm: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>devices parm</td>
<td>devices parm: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>num of devices</td>
<td>num of devices: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>vary results</td>
<td>vary results: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>return code</td>
<td>return code: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>reason code</td>
<td>reason code: RS-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

**OPERATION=operation parm**

Specifies the address or name of a required 16-byte input area in which you place the VARY device service portion of the initialized VDEV data area, mapped by the IEEZB833 mapping macro. The VARY device service portion is the VDEV control block.

**devices parm**

Specifies the address or name of a required variable-length area in which you place the IEEVARYD device array entries. Each device entry is mapped by the VDEVARR DSECT of the IEEZB833 mapping macro.

**num of devices**

Specifies the name or address of an optional fullword input area where you specify the number of devices in the device array specified on the DEVICES parameter.

The default is 1.

**vary results**

Specifies the name or address of an optional array (mapped by the IEEZB834 macro) that contains results for the VARY operation on each device specified in the DEVICES parameter.

**caller id**

Specifies the name or address of an optional 8-character input area into which you place the name of the caller associated with the VARY operation. If CALLERID is specified, then MVS inserts the identifier into the one or more...
messages that MVS issues to hardcopy, which indicate that a device was brought online or taken offline. For example, if a program uses IEEVARYD to VARY device 205 online and specifies XYZ as the CALLERID, MVS issues the following message to hardcopy:

```
IEE302I 0205 ONLINE BY XYZ
```

If the program did not specify the caller ID, the message to hardcopy would be:

```
IEE302I 0205 ONLINE
```

,**RETCODE=return code**

Specifies an optional fullword output area into which IEEVARYD will copy a return code from GPR 15.

,**RSNCODE=reason code**

Specifies an optional fullword output area into which IEEVARYD will copy a reason code from GPR 0.

**ABEND codes**

The IEEVARYD macro abnormally terminates with abend code 077-003C if the ENQs are not released before invoking IEEVARYD.

**Return and reason codes**

When IEEVARYD returns control to your program, GPR 15 contains a return code and GPR 0 contains a reason code. If you specified the RETCODE or RSNCODE parameters, those areas will also contain a return code and reason code, respectively.

The following table identifies return code and reason code combinations, tells what each means, and recommends an action that you need to take.

**Table 70. Return and Reason Codes for the IEEVARYD Macro**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00          | 00          | **Meaning**: Processing completed successfully.  
**Action**: No action needed. |
| 04          | None.       | **Meaning**: The operation was performed against all specified devices, but MVS set the VDRSARR RETCODE field in the IEEZB834 mapping macro to a non-zero value for at least one device in the device array.  
**Action**: In the RESULTS area, check the VDRSARR RETCODE values for each device to determine the result of the operation on each device. |
| 08          | 01          | **Meaning**: Program error. MVS could not access the caller’s parameter list.  
**Action**: Ensure that you have specified the correct parameter list area on the execute form of the macro. |
| 08          | 02          | **Meaning**: Program error. MVS could not access the storage area specified on the DEVICES parameter.  
**Action**: Ensure that the DEVICES parameter correctly specifies the name or address of storage that contains the device array. The entire device array must reside in storage that is accessible to the program invoking the IEEVARYD macro. |
### IEEVARYD Macro

#### Table 70. Return and Reason Codes for the IEEVARYD Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | 03          | **Meaning:** Program error. MVS could not access the storage area specified on the OPERATION parameter.  
**Action:** Ensure that the OPERATION parameter correctly specifies the name or address of storage that contains the VDEV control block. The VDEV control block must reside in storage that is accessible to the program invoking the IEEVARYD macro. |
|             | 04          | **Meaning:** Program error. MVS could not access the storage area specified on the RESULTS parameter.  
**Action:** Ensure that the RESULTS parameter correctly specifies the name or address of storage that contains the device results array. The entire device results array must reside in storage that can be updated by the program invoking the IEEVARYD macro. |
| 08          | 05          | **Meaning:** Program error. The input parameter list includes an invalid combination of VARY command keywords. For example:  
- Both ONLINE and OFFLINE  
- Neither ONLINE nor OFFLINE  
- Both OFFLINE and RESET  
- AUTOSWITCH,ON or AUTOSWITCH,OFF with any other parameters.  
**Action:** Ensure that the OPERATION parameter identifies a VDEV data area that specifies flags in the VDEV control block that are valid according to guidelines of the VARY command. For information about the VARY command, see [z/OS MVS System Commands](#). |
| 08          | 06          | **Meaning:** Program error. The input parameter list includes an invalid combination of VARY COMMAND options. For example:  
- KEEP_OFFLINE with ONLINE  
**Action:** Ensure that the OPERATION parameter identifies a VDEV data area that specifies flags in the VDEV control block that are valid according to guidelines of the VARY command. For information about the VARY command, see [z/OS MVS System Commands](#). |
| 08          | 07          | **Meaning:** Program error. The NUMDEVS value is not valid.  
**Action:** Change NUMDEVS to a valid value. Valid values are 1 to 65536. |
| 0C          | 00          | MVS was temporarily unable to process the requested operation. The caller requested the VDEV_DO_NOT_WAIT_FOR_ENQ option, and the IEEVARYD service was unable to obtain the SYSIEFSD.Q4 resource in a reasonable amount of time. The operation might be successful if retried at a later time. |
| 0C          | 04          | MVS was temporarily unable to process the requested operation. The caller requested the VDEV_DO_NOT_WAIT_FOR_ENQ option, and the IEEVARYD service was unable to obtain the SYSIEFSD.VARYDEV resource in a reasonable amount of time. The operation might be successful if retried at a later time. |
### IEEVARYD macro

#### Table 70. Return and Reason Codes for the IEEVARYD Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 10          | None        | **Meaning:** System error. Some devices were processed, and some were not processed.  
**Action:** Check the RESULTS area to determine which devices were processed. Record this code and supply it to IBM support personnel. |

#### Examples

Example 1 shows how you define tape devices 200 and 300, both in a varied-offline state, as automatically switchable. The second example then varies the devices online.

**Example 1**

Use the IEEVARYD macro to define devices 200 and 300 as automatically switchable. The example includes steps to:

1. Initialize the IEEVARYD input for the operation by setting the appropriate keyword and option flags.
2. Initialize the IEEVARYD device array entries with the device number of each device upon which the operation is to be performed.
3. Issue the execute form of the IEEVARYD, specifying the VARY device service input, IEEVARYD device array, VARY device service results, and the IEEVARYD parameter list which was defined when the list form of IEEVARYD was issued.
4. Examine the return code returned in register 15 to determine the overall result of the operation.
5. Use the IEEZB834 mapping macro to determine the results of the operation for each device in the IEEVARYD device array.
6. Free the storage for the IEEVARYD input, IEEVARYD device array, and VARY device service results.

```
* Issue the list form of the IEEVARYD to define the IEEVARYD macro parameters.
* Include the IEEZB833 and IEEZB834 mapping macros in the program declarations.
* IEEZB833 maps the IEEVARYD input including the IEEVARYD device array (VDEVARR). IEEZB834 maps the VARY device service results.
* Obtain storage for the IEEVARYD input, and the VARY device service results. Obtain storage for an IEEVARYD device array for each device affected by a single invocation of IEEVARYD. (This example uses two IEEVARYD device arrays.) The address of the storage area is in R1.
* Initialize the IEEVARYD input:
  * LR R2,R1 Address of storage
  * LA R4,0 Set to zero for MVCL
  * LA R5,0 Set to zero for MVCL
  * MVCL R2,R4 Clear storage
  * USING VDEV,R2 Obtain addressability
  * L R0,CBID Load identifier
  * ST R0,VDEV_ID Initialize the identifier
```
### IEEVARYD macro

```
MVI VDEV_VERSION, VDEV_VERN Initialize the version number
OI VDEV_KEYWORDS1, VDEV_AUTOSWITCH Initialize the operation to AUTOSWITCH
* OI VDEV_KEYWORDS2, VDEV_ON Indicate to turn it ON
LA R0, VDEV_LENGTH Length of the IEEVARYD input
LR R4, R2 Address of the IEEVARYD input
ALR R4, R0 Address of storage immediately following the IEEVARYD input
* LR R5, R4 Save address of the IEEVARYD device array
*
* Initialize the first IEEVARYD device array entry
* USING VDEVARR, R4 Obtain addressability
LA R0, 200
STH R0, VDEVARR_DEVN Initialize the device number (200)
*
* Initialize the second IEEVARYD device array entry
* LA R0, VDEVARR_LENGTH Length of IEEVARYD device array entry
ALR R4, R0 Obtain addressability to next entry
LA R0, 300
STH R0, VDEVARR_DEVN Initialize the device number (300) following the IEEVARYD input
******************************************************************************
* Get address of IEEVARYD Results
******************************************************************************
ALR R4, R0 Address of VDRSARR
******************************************************************************
* Issue the execute form of IEEVARYD to vary the devices online
******************************************************************************
IEEVARYD OPERATION=(R2), DEVICES=(R5), NUMDEVS=#DEVS, X
RESULTS=(R4), CALLERID=VDEVICES_ID, MF=(E, IEEVARYL)
* Determine if the operation was not performed because of an error in
* the parameters (Register 15 = 8)
* LA R0, 8
CR R15, R0
BE FREEVDEV If the parameters are in error, free
* the storage for the vary device array
* USING VDRSARR, R4 Obtain addressability to RESULTS
LA R7, 1 Initialize counter for loop
*
* Perform the following loop for each IEEVARYD device array entry to
* determine the results of the operation against each device
*
LOOP DS 0H All entries processed?
C R7, #DEVS
BH ENLOOP
TM VDRSARR_OUTPUT_FLAGS1, VDRSARR_OUTPUT_VALID Determine if output was returned for the device
* BZ ITERATE No output for the device, so iterate
L R9, VDRSARR_RETCODE Get return code for the device
LA R0, VDRSARR_ALREADY_OK Set register 0 to
* CR R9, R0 Determine if the operation was
* successful against the device
* BH BADRETC
GOODRETC DS 0H The operation was successful against
* the device
B CHECKMSG
```
IEEVARYD macro

BADRETC DS 0H The operation was not successful
* against the device
CHECKMSG DS 0H Determine if a message was returned
* for the device
  TM VDRSARR_OUTPUT_FLAGS1,VDRSARR_MSG_RETURNED
  BZ NOMSG
  MSG DS 0H A message was returned for the device
  B ITEATE
  NOMSG DS 0H A message was not returned for the
device
  ITERATE DS 0H Prepare for the next iteration
  LA R0,1
  ALR R7,R0 Increment loop counter
  LA R8,VDRSARR_LENGTH Length of IEEVARYD device array entry
  ALR R4,R0 Obtain addressability to next entry
  B LOOP Iterate
ENDLOOP DS 0H End of loop
*
* Release the storage for the IEEVARYD input, two
* IEEVARYD device array entries, and IEEVARYD Results
*
FREEVDEV DS 0H
*******************************************************************************
** Declarations
*******************************************************************************
CBID DC C'VDEV' Control block identifier for the
* IEEVARYD input
VDEVICES_ID DC C'VDEVICES' Caller identifier for the IEEVARYD
* Input
@DATA DS 0H
@DATD DSECT
DS 0F
#DEVS DS F
*******************************************************************************
* Issue the list form of IEEVARYD to define the parameter list
*
*******************************************************************************
IEEVARYD MF=(L,IEEVARYL)
@ENDDATD DS 0X
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
*******************************************************************************
* Include IEEZB833 to define the IEEVARYD input (VDEV) and
* IEEVARYD device array entry (VDEVARR). Include IEEZB834
* to define the IEEVARYD Results (VDRSARR).
Example 2

The following example illustrates how a program can use IEEVARYD to vary devices 200 and 300 online. The example includes steps to:

1. Issue the list form of IEEVARYD to define the IEEVARYD macro parameter list.
2. Include the IEEZB833 and IEEZB834 mapping macros in the program.
   IEEZB833 maps the IEEVARYD input including the IEEVARYD device array (VDEVARR). IEEZB834 maps the VARY device service results.
3. Obtain storage for the IEEVARYD input, IEEVARYD device array, and VARY device service results. An IEEVARYD device array entry is required for each device affected by a single invocation of IEEVARYD.
4. Initialize the IEEVARYD input for the operation by setting the appropriate keyword and option flags.
5. Initialize the IEEVARYD device array entries with the device number of each device upon which the operation is to be performed.
6. Issue the execute form of IEEVARYD, specifying the VARY device service input, IEEVARYD device array, VARY device service results, and the IEEVARYD parameter list which was defined when the list form of IEEVARYD was issued.
7. Examine the return code returned in register 15 to determine the overall result of the operation.
8. Use the IEEZB834 mapping macro to determine the results of the operation for each device in the IEEVARYD device array.
9. Free the storage for the IEEVARYD input, IEEVARYD device array, and VARY device service results.

**********************************************************************

* Obtain storage for the IEEVARYD input, two IEEVARYD device array
* Entries, and IEEVARYD Results
* The address of the storage area is in R1
*
* Initialize the IEEVARYD input
*
   LR  R2,R1     Address of storage
   LA  R4,0      Set to zero for MVCL
   LA  R5,0      Set to zero for MVCL
   MVCL R2,R4    Clear storage
   USING VDEV,R2 Obtain addressability
   L   R0,CBID   Load identifier
   ST  R0,VDEV_ID Initialize the identifier
   MVI VDEV_VERSION,VDEV_VERN Initialize the version number
   OI  VDEV_KEYWORDS1,VDEV_ONLINE Initialize the operation to
       ONLINE
   LA  R0,VDEV_LENGTH Length of the IEEVARYD input
   LR  R4,R2     Address of the IEEVARYD input
   ALR R4,R0     Address of storage immediately
* following the IEEVARYD input
   LR  R5,R4     Save address of the
* IEEVARYD device array
*
* Initialize the first IEEVARYD device array entry
*
   USING VDEVARR,R4 Obtain addressability
   LA  R0,200    Address of the device number (200)
   STH R0,VDEVARR_DEVN Initialize the device number (200)

**********************************************************************
**IEEVARYD macro**

* Initialize the second IEEVARYD device array entry

* \[
\text{LA R0, VDEVARR\_LENGTH} \quad \text{Length of IEEVARYD device array entry}
\]

* \[
\text{ALR R4, R0} \quad \text{Obtain addressability to next entry}
\]

* \[
\text{LA R0, 300} \quad \text{Initialize the device number (300)}
\]

* following the IEEVARYD input

******************************************************************************

* Get address of IEEVARYD Results

******************************************************************************

* \[
\text{ALR R4, R0} \quad \text{Address of VDRSARR}
\]

******************************************************************************

* Issue the execute form of IEEVARYD to vary the devices online

******************************************************************************

* \[
\text{IEEVARYD OPERATION=(R2), DEVICES=(R5), NUMDEVS=#DEVS, X}
\]

* \[
\text{RESULTS=(R4), CALLERID=VDEVICES\_ID, MF=(E, IEEVARYL)}
\]

* Determine if the operation was not performed because of an error in
the parameters (Register 15 = 8)

* \[
\text{LA R0, 8}
\]

* \[
\text{CR R15, R0}
\]

* BE FREEVDEV \quad \text{If the parameters are in error, free}

* \text{the storage for the vary device array}

* \text{header and vary device array entries}

* \[
\text{USING VDRSARR, R4} \quad \text{Obtain addressability to RESULTS}
\]

* LA R7, 1 \quad \text{Initialize counter for loop}

* Perform the following loop for each IEEVARYD device array entry to
determine the results of the operation against each device

* \[
\text{LOOP DS 0H}
\]

* \[
\text{C R7, #DEVS} \quad \text{All entries processed?}
\]

* \[
\text{BH ENDL} \quad \text{ENDLOOP}
\]

* \[
\text{TM VDRSARR\_OUTPUT\_FLAGS1, VDRSARR\_OUTPUT\_VALID} \quad \text{Determine if}
\]

* \text{output was returned for the device}

* \[
\text{BZ ITERATE} \quad \text{No output for the device, so iterate}
\]

* \[
\text{L R9, VDRSARR\_RETCODE} \quad \text{Get return code for the device}
\]

* \[
\text{LA R0, VDRSARR\_ONLINE\_WITH\_REST} \quad \text{Set register 0 to}
\]

* \[
\text{VDRSARR\_ONLINE\_WITH\_REST}
\]

* \[
\text{CR R9, R0} \quad \text{Determine if the operation was}
\]

* \text{successful against the device}

* \[
\text{(VDRSARR\_RETCODE <= VDRSARR\_ONLINE\_WITH\_REST)}
\]

* \[
\text{BH BADRETC} \quad \text{The operation was not successful}
\]

* \text{against the device}

* \[
\text{GOODRETC DS 0H} \quad \text{The operation was successful against}
\]

* \text{the device}

* \[
\text{BADRETC DS 0H} \quad \text{The operation was not successful}
\]

* against the device

* \[
\text{CHECKMSG DS 0H} \quad \text{Determine if a message was returned}
\]

* for the device

* \[
\text{TM VDRSARR\_OUTPUT\_FLAGS1, VDRSARR\_MSG\_RETURNED}
\]

* \[
\text{BZ NOMSG}
\]

* \[
\text{MSG DS 0H} \quad \text{A message was returned for the device}
\]

* \[
\text{B ITERATE}
\]

* \[
\text{NOMSG DS 0H} \quad \text{A message was not returned for the}
\]

* \text{device}

* \[
\text{ITERATE DS 0H} \quad \text{Prepare for the next iteration}
\]

* \[
\text{LA R0, 1}
\]

* \[
\text{ALR R7, R0} \quad \text{Increment loop counter}
\]

* \[
\text{LA R0, VDEVARR\_LENGTH} \quad \text{Length of IEEVARYD device array entry}
\]

* \[
\text{ALR R4, R0} \quad \text{Obtain addressability to next entry}
\]
IEEVARYD - List form

Use the list form of the IEEVARYD macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters.

The list form of the IEEVARYD macro is written as follows:

IEEZB833
IEEZB834
END VDEVICES
### IEEVARYD macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEEVARYD.</td>
</tr>
<tr>
<td>IEEVARYD</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEEVARYD.</td>
</tr>
</tbody>
</table>

| MF=(L,list addr)            | list addr: symbol.                             |
| MF=(L,list addr,attr)       | attr: 1- to 60-character input string.         |
| MF=(L,list addr,0D)         | Default: 0D                                    |

The parameters are explained as follows:

- **MF=(L,list addr)**
- **MF=(L,list addr,attr)**
- **MF=(L,list addr,0D)**

Specifies the list form of the IEEVARYD macro.

- **list addr** is the name of a storage area to contain the parameters.

- **attr** is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

### IEEVARYD - Execute form

Use the execute form of the IEEVARYD macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The execute form of the IEEVARYD macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEEVARYD.</td>
</tr>
<tr>
<td>IEEVARYD</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEEVARYD.</td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPERATION=operation parm</td>
<td>operation parm: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,DEVICES=devices parm</td>
<td>devices parm: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,NUMDEV= num of devices</td>
<td>num of devices: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RESULTS= vary results</td>
<td>vary results: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,CALLERID= caller id</td>
<td>caller id: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RETCODE= return code</td>
<td>return code: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE= reason code</td>
<td>reason code: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td>list addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td>Default: COMPLETE</td>
</tr>
</tbody>
</table>

The parameters are explained under the standard form of the IEEVARYD macro with the following exceptions:

- ,MF=(E,list addr)
- ,MF=(E,list addr,COMPLETE)

Specifies the execute form of the IEEVARYD macro.

`list addr` specifies the area that the system uses to store the parameters.

**COMPLETE**, which is the default, specifies that the system is to check for required parameters and supply optional parameters that are not specified.
Chapter 77. IEFPPSCN — Scan the program properties table

Description
The IEFPPSCN macro provides a way to retrieve information, for report generation, about programs that are listed in the program properties table (PPT). IEFPPSCN allows the calling program to scan each entry in the PPT or to search the PPT for a specific program.

The installation controls what programs are listed in the PPT. An installation can specify a list of programs that require special attributes by using the SCHEDxx parmlib member with the PPT statement. The system then creates entries for these programs in the PPT. See z/OS MVS Initialization and Tuning Reference for information about using the SCHEDxx parmlib member.

The contents of the PPT can be dynamically changed through the SET command. Using the IEFPPSCN macro to retrieve information from the PPT prevents the system from dynamically updating the PPT while you are scanning it. If you scan the PPT without using IEFPPSCN, and the system updates the PPT while you are scanning it, your program will abnormally end.

Environment
Requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state and PSW key 0</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space</td>
</tr>
</tbody>
</table>

Programming requirements
The calling program must include the following mapping macros:

- CVT
- IEFJESCT
- IEFZB610

Restrictions
None.

Register information
After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the caller issued the macro. Therefore, if the caller depends on these
IEFPSCPSCN macro

registers containing the same value before and after issuing the macro, the caller
must save these registers before issuing the macro and restore them after the
system returns control.

When control is returned to the calling program the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

Performance implications

None.

Syntax

The standard form of the IEFPPSCPSCN macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFPPSCPSCN.</td>
</tr>
<tr>
<td>IEFPPSCPSCN</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFPPSCPSCN.</td>
</tr>
<tr>
<td>REQUEST=RETRIEVE</td>
<td></td>
</tr>
<tr>
<td>REQUEST=NEXT</td>
<td></td>
</tr>
<tr>
<td>REQUEST=END</td>
<td></td>
</tr>
<tr>
<td>,PPTINFO=ppt_info</td>
<td>ppt_info: RX-type address or register (2) - (12). Required for REQUEST=RETRIEVE and REQUEST=NEXT. Not valid for REQUEST=END.</td>
</tr>
<tr>
<td>,PROGRAM=program_name</td>
<td>program_name: RX-type address or register (2) - (12). Required for REQUEST=RETRIEVE. Not valid for REQUEST=NEXT or REQUEST=END.</td>
</tr>
<tr>
<td>,TOKEN=token</td>
<td>token: RX-type address or register (2) - (12). Required for REQUEST=NEXT and REQUEST=END. Not valid for REQUEST=RETRIEVE.</td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

REQUEST=RETRIEVE
REQUEST=NEXT
REQUEST=END

The required parameter that specifies what kind of request the caller is making.

Specify REQUEST=RETRIEVE to request information about a specific program. You must specify the name of the program on the PROGRAM parameter. You must also specify the PPTINFO parameter. Do not specify the TOKEN parameter.

If you want to scan all program entries sequentially, use the REQUEST=NEXT parameter together with the REQUEST=END parameter. Each time you specify REQUEST=NEXT, the system retrieves information about the next program entry. The first time you specify REQUEST=NEXT, you must put zero in the field you provide on the TOKEN parameter. On return to the caller, the system places a value in this field. After the first call, when you specify REQUEST=NEXT you must specify TOKEN and supply the value provided by the system on the previous call. With REQUEST=NEXT, you must also specify the PPTINFO parameter. Do not specify the PROGRAM parameter.

When control returns to the calling program with a return code of 4 in GPR 15, you have reached the end of the table and must then specify REQUEST=END. If you use REQUEST=NEXT and do not specify REQUEST=END, the system might not free common storage that could have been freed.

When you specify REQUEST=END, you must also specify the TOKEN parameter, supplying the value returned on the last REQUEST=NEXT. Do not specify the PROGRAM or PPTINFO parameters.

PPTINFO=ppt_info

Specifies the area provided by the caller to contain the requested program information. The caller must provide this area as follows:

• The length of the area must be the length of the PPTENTRY field of the PPT plus two bytes. The PPT is mapped by the IEFZB610 mapping macro. See PPT in z/OS MVS Data Areas in the z/OS Internet library [http://www.ibm.com/systems/z/os/zos/bkserv/] for the PPT mapping.
• Initialize the first two bytes of the area to the length of the PPTENTRY field.
• Define the length of the remainder of the area to be equal to the length of the PPTENTRY field.

Upon return, the system places the length of the requested PPTENTRY in the first two bytes of the area, and places the requested PPTENTRY itself in the remainder of the area.

This area must have a storage key that matches the PSW key of the issuer of IEFPPSCN. If the area provided is too small, the information is truncated. PPTINFO is required for REQUEST=RETRIEVE and REQUEST=NEXT. Do not specify PPTINFO with REQUEST=END.
IEFPPSCN macro

,,PROGRAM=program_name
  Specifies the 8-character field containing the name of the program you want to
  retrieve when you specify REQUEST=RETRIEVE. If the program name is less
  than 8 characters, left justify the name and pad on the right with blanks.

  Do not specify PROGRAM with REQUEST=NEXT or with REQUEST=END.

,,TOKEN=token
  Specifies the 4-byte field to contain the token that the system returns when you
  specify REQUEST=NEXT. Set the value of the token to zero before the first call.
  On subsequent calls made with REQUEST=NEXT or REQUEST=END, use the
  value of the token returned by the system on the previous REQUEST=NEXT.

  Do not specify TOKEN with REQUEST=RETRIEVE.

,,RETCODE=rc
  Specifies the location where the system is to store the return code. The return
  code is also in GPR 15.

Return codes

When control returns from IEFPPSCN, GPR 15 (and rc, if you coded RETCODE)
contains one of the following return codes:

Table 71. Return Codes for the IEFPPSCN Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: Program found.</td>
</tr>
<tr>
<td></td>
<td>Action: No action required.</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: End of table.</td>
</tr>
<tr>
<td></td>
<td>Action: Issue IEFPPSCN with REQUEST=END.</td>
</tr>
<tr>
<td>08</td>
<td>Meaning: The program name you specified is not listed in the PPT, indicating that the installation did not specify any special attributes for that program.</td>
</tr>
<tr>
<td></td>
<td>Action: No action required.</td>
</tr>
<tr>
<td>0C</td>
<td>Meaning: The token passed was not the token created by this macro.</td>
</tr>
<tr>
<td></td>
<td>Action: Check that the application is coded to pass the correct token.</td>
</tr>
<tr>
<td>10</td>
<td>Meaning: Request not valid.</td>
</tr>
<tr>
<td></td>
<td>Action: Check that you did not change the expanded assembler code.</td>
</tr>
<tr>
<td>14</td>
<td>Meaning: System error. The system was not able to obtain the required storage. Your program might issue a message indicating incomplete scanning of the PPT.</td>
</tr>
<tr>
<td></td>
<td>Action: Reissue the request. If the error persists, contact your IBM support personnel.</td>
</tr>
</tbody>
</table>

Example

Sequentially scan the PPT and write the program name of each entry to the
console. In your own code, you might wish to format the non-EBCDIC portions of
the PPT entry for inclusion in the WTO.

The code in this example is nonreentrant. The caller is APF-authorized, and is
initially in problem state with PSW key 8. The caller changes to supervisor state
with PSW key 0 before issuing IEFPPSCN, and returns to problem state with PSW
key 8 on completion of processing.

Note: This example is in SYS1.SAMPLIB in the member SHOWPPT.
**IEFPPSCN macro**

```
TITLE 'SHOWPPT - Show all entries in Current PPT'
SHOWPPT CSECT Module entry point
SHOWPPT AMODE 31
SHOWPPT RMODE ANY
*
* Body of nonreentrant module which prints program names in the PPT
*
STM 14,12,12(13) Standard module linkage
LR 12,15
* USING SHOWPPT,12
ST 13,SAVEAREA+4
LR 2,13
LA 13,SAVEAREA
ST 13,8(2)
*
MODESET MODE=SUP,KEY=ZERO Need supervisor state, key 0
*
* Set up for looking at PPT entries
*
LA 2,PPTENT Point to copy of PPT entry
USING PPTENTRY,2 Set up addressability
XC SCNTOKEN,SCNTOKEN Clear token
*
LOOP DS 0H Loop getting PPT entries
IEFPPSCN REQUEST=NEXT,TOKEN=SCNTOKEN,PPTINFO=PPTE
LTR 15,15 Check whether entry was returned
BNZ ENDLOOP
MVC TEXT2(8),PPTNAME Copy program name to message
WTO TEXT=ENTRYTXT Get next entry, if any
*
ENDLOOP DS 0H Finished with PPT
IEFPPSCN REQUEST=END,TOKEN=SCNTOKEN
MODESET MODE=PROB,KEY=NZERO Return to problem state
*
* Return to the calling program with the return code last passed by
* IEFPPSCN.
*
L 13,SAVEAREA+4 Return linkage
L 14,12(13)
LM 0,12,20(13) 
BR 14
*
**** Local storage definitions ****
SCNTOKEN DC A(0) PPT scan token
PPTE DC AL2(L'PPTENTRY) Length of a PPT entry
PPTENT DS CL(L'PPTENTRY) PPT entry return area
SAVEAREA DC 18F'0'
*
* The following areas are used to print the program name within the
* PPT. Additional formatting is required to make all the
* information readable.
*
ENTRYTXT DS 0F Area for printing
TEXTL DC H'32' Message length
TEXT1 DC C'SHOWPPT: Program Name = ' Constant portion of message
TEXT2 DS CL8 Variable portion (program name)
*
* The following mapping macros are required for the IEFPPSCN macro.
*
CVT DSECT=YES
IEFJESCT
IEFZB610
END SHOWPPT End of SHOWPPT
```
IEFPPSCN macro

IEFPPSCN - List form

Use the list form of the IEFPPSCN macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters.

Syntax

The list form of the IEFPPSCN macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFPPSCN.</td>
</tr>
<tr>
<td>IEFPPSCN</td>
<td>One or more blanks must follow IEFPPSCN.</td>
</tr>
<tr>
<td>,MF=(L,cnt1)</td>
<td>cnt1: Symbol.</td>
</tr>
<tr>
<td>,MF=(L,cnt1,attr)</td>
<td>attr: 1- to 60-character input string.</td>
</tr>
<tr>
<td>,MF=(L,cnt1,0D)</td>
<td>Default: 0D.</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained under the standard form of the IEFPPSCN macro with the following exception:

,MF=(L,cnt1)
,MF=(L,cnt1,attr)
,MF=(L,cnt1,0D)

Specifies the list form of the macro.

cnt1 is the name of a storage area for the parameter list.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

IEFPPSCN - Execute form

Use the execute form of the IEFPPSCN macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.
# Syntax

The execute form of the IEFPPSCN macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFPPSCN.</td>
</tr>
<tr>
<td>IEFPPSCN</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFPPSCN.</td>
</tr>
<tr>
<td>REQUEST=RETRIEVE</td>
<td></td>
</tr>
<tr>
<td>REQUEST=NEXT</td>
<td></td>
</tr>
<tr>
<td>REQUEST=END</td>
<td></td>
</tr>
<tr>
<td>,PPTINFO= ppt_info</td>
<td>ppt_info: RX-type address or register (2) - (12). Required for REQUEST=RETRIEVE and REQUEST=NEXT. Not valid for REQUEST=END.</td>
</tr>
<tr>
<td>,PROGRAM= program_name</td>
<td>program_name: RX-type address or register (2) - (12). Required for REQUEST=RETRIEVE. Not valid for REQUEST=NEXT or REQUEST=END.</td>
</tr>
<tr>
<td>,TOKEN= token</td>
<td>token: RX-type address or register (2) - (12). Required for REQUEST=NEXT and REQUEST=END. Not valid for REQUEST=RETRIEVE.</td>
</tr>
<tr>
<td>,RETCODE= rc</td>
<td>rc: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=(E,cntl)</td>
<td>cntl: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=(E,cntl,COMPLETE)</td>
<td>Default: COMPLETE</td>
</tr>
</tbody>
</table>

# Parameters

The parameters are explained under the standard form of the IEFPPSCN macro with the following exception:

- ,MF=(E,cntl)
- ,MF=(E,cntl,COMPLETE)  
  Specifies the execute form of the macro.
  
  cntl is the name of a storage area for the parameter list.
**IEFPPSCN macro**

COMPLETE specifies that the system is to check the macro parameter syntax and supply defaults on parameters that you do not use. COMPLETE is the default.
Chapter 78. IEFQMREQ — Invoke SWA manager in move mode

Description

Use this macro to read information from the SWA into a buffer that you provide, or to write information from a buffer into the SWA. [z/OS MVS Programming: Authorized Assembler Services Guide] describes how to use the IEFQMREQ macro.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state, and any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the caller’s primary address space</td>
</tr>
</tbody>
</table>

Programming requirements

The caller must include the following mapping macros:

- CVT
- IEFJESCT
- IEFQMIDS
- IEFQMNGR
- IEFZB506

Provide input to the IEFQMREQ macro through the external parameter area (EPAM), mapped by IEFZB506, and the queue manager parameter area (QMPA), mapped by IEFQMNGR.

Restrictions

None.

Input register information

On input to the macro, general purpose register (GPR) 1 must contain the address of the QMPA, and GPR 13 must contain the address of a standard 18-word save area.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>When control returns from IEFQMREQ, unchanged.</td>
</tr>
</tbody>
</table>
IEFQMREQ macro

When control does not return from IEFQMREQ, the address of a 16-byte area containing:

**Bytes 1-4**
Address of the QMPA

**Bytes 5-12**
Not an intended programming interface; record this information and provide it to the appropriate IBM support personnel.

**Bytes 13-16**
Address of the failing EPA

1 When control returns from IEFQMREQ, used as a work register by the system.
When control does not return from IEFQMREQ, abend code 0B0.

2-14 Unchanged

15 Return code, when control returns from IEFQMREQ.
Reason code associated with the abend, when control does not return from IEFQMREQ.

**Syntax**

The IEFQMREQ macro, which has no parameters, is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name:</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFQMREQ.</td>
</tr>
<tr>
<td>IEFQMREQ</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFQMREQ.</td>
</tr>
</tbody>
</table>

**Parameters**

For information about initializing the parameter areas for IEFQMREQ, see [z/OS MVS Programming: Authorized Assembler Services Guide](https://www.ibm.com/support/docview.wss?uid=swg27046533).

**ABEND codes**

The caller might encounter abend code X'0B0'.

**Return and reason codes**

The hexadecimal return code is in GPR 15. When control returns from IEFQMREQ, the return codes have the following meanings.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td><strong>Meaning:</strong> The IEFQMREQ service was successful.</td>
</tr>
</tbody>
</table>
### IEFQMREQ Macro

#### Table 72. Return Codes for the IEFQMREQ Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td><strong>Meaning:</strong> The system could not obtain the storage necessary to carry out the request.</td>
</tr>
</tbody>
</table>

When control does not return from IEFQMREQ, GPR 15 contains a hexadecimal reason code associated with system abend code 0B0. The reason codes have the following meanings.

#### Table 73. Reason Codes for the IEFQMREQ Macro

<table>
<thead>
<tr>
<th>Hexadecimal Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td><strong>Meaning:</strong> The function you requested was not valid.</td>
</tr>
<tr>
<td>08</td>
<td><strong>Meaning:</strong> The SVA in the SWA prefix was not valid.</td>
</tr>
<tr>
<td>0C</td>
<td><strong>Meaning:</strong> You attempted to read a block that was not yet written.</td>
</tr>
<tr>
<td>10</td>
<td><strong>Meaning:</strong> The length of an SWA block was not valid.</td>
</tr>
<tr>
<td>14</td>
<td><strong>Meaning:</strong> The count field was not valid.</td>
</tr>
<tr>
<td>1C</td>
<td><strong>Meaning:</strong> The block ID was not valid.</td>
</tr>
<tr>
<td>24</td>
<td><strong>Meaning:</strong> The SVA does not correspond to any virtual address.</td>
</tr>
</tbody>
</table>
IEFQMREQ macro
Chapter 79. IEFSJSYM — JCL symbol service

Description

The IEFSJSYM JCL symbol service provides JCL symbol information from the submitted JCL to the program running under the submitted JCL. The JCL symbol service performs the following functions:

**REQUEST=GETALL**
Returns all of the JCL symbols and values for the job step in the area provided by the caller, and is mapped by the IEFSJSYD macro.

**REQUEST=GETBYNAME**
Returns symbol values for the symbol names provided by the caller by the SymListArray parameter.

To be visible to the program, the symbols must have been either exported prior to the job step or provided by the submitter. The symbols are returned without a leading ampersand character (&).

The following information is described once at the beginning of the IEFSJSYM macro description:

- Environment
- Programming requirements
- Restrictions
- Input register information
- Output register information
- Performance implications

Following the descriptions of the standard forms of all requests are:

- Abend codes
- Return and reason codes
- Examples

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state with any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit or 64-bit</td>
</tr>
</tbody>
</table>

If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.

- ASC mode: Primary or Access register (AR)
- Interrupt status: Enabled for I/O and external interrupts
- Locks: No locks held
- Control parameters: Control parameters must be in the primary address space
IEFSJSYM macro

Programming requirements

REQUEST=GETBYNAME

The caller must provide a list of valid symbol names. Invalid symbol names or symbols that were not exported will have a null symbol value and a symbol value length of zero. A return code of 4 will be returned to indicate that not all symbols were processed successfully.

IEFSJSYD macro

To map data returned by an IEFSJSYM request.

Restrictions

This service cannot be used reliably until the job has begun execution. Invoking the service before the first job step has started executing is not supported (for example, in exits such as IEFUJI that are invoked before the first job step has started executing).

When using the returned symbol values, the value of the symbol returned will be the last value set prior to or within the current job step (EXEC PGM=statement).

Input register information

There are no input register requirements for issuing the IEFSJSYM macro.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.
The IEFSJSYM macro with the REQUEST parameter produces a DSECT that maps the format of the function routine input table.

## Syntax

The syntax of the IEFSJSYM macro with REQUEST= is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSJSYM.</td>
</tr>
<tr>
<td>IEFSJSYM</td>
<td>One or more blanks must follow IEFSJSYM.</td>
</tr>
<tr>
<td>REQUEST=GETALL</td>
<td>Either REQUEST=GETALL or REQUEST=GETBYNAME is required.</td>
</tr>
<tr>
<td>REQUEST=GETBYNAME</td>
<td>Required for REQUEST=GETBYNAME only.</td>
</tr>
<tr>
<td>SYMLISTARRAY=symlistarray</td>
<td>Required for REQUEST=GETBYNAME only.</td>
</tr>
<tr>
<td>NUMENTRIES=numentries</td>
<td>Required for REQUEST=GETBYNAME only.</td>
</tr>
<tr>
<td>SYMBAREA=symbarea</td>
<td></td>
</tr>
<tr>
<td>SYMBAREALEN=symbarealen</td>
<td></td>
</tr>
<tr>
<td>DIAGDATA=diagdata</td>
<td></td>
</tr>
<tr>
<td>RETCODE=retcode</td>
<td></td>
</tr>
<tr>
<td>RSNCODE=rsncoolde</td>
<td></td>
</tr>
<tr>
<td>PLISTVER=IMPLIED_VERSION</td>
<td>IMPLIED_VERSION is the default value.</td>
</tr>
<tr>
<td>PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>PLISTVER=0</td>
<td></td>
</tr>
<tr>
<td>MF=S</td>
<td>S is the default value.</td>
</tr>
<tr>
<td>MF=(L,list addr,0D)</td>
<td>0D is the default value.</td>
</tr>
<tr>
<td>MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>MF=(E,list addr,COMPLETE)</td>
<td>COMPLETE is the default value.</td>
</tr>
<tr>
<td>MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
<tr>
<td>MF=(M,list addr,COMPLETE)</td>
<td>COMPLETE is the default value.</td>
</tr>
<tr>
<td>MF=(M,list addr,NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>
IEFSJSYM macro

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEFSJSYM macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,DIAGDATA=diagdata
A required output parameter that specifies an area for service to return additional information. To code this parameter, specify an RS-type address, or address in register (2)-(12), of a 16-character field.

, MF=S,MF=(L,list addr)
, MF=(L,list addr,attr)
, MF=(L,list addr,0D)
, MF=(E,list addr)
, MF=(E,list addr,COMPLETE)
, MF=(E,list addr,NOCHECK)
, MF=(M,list addr)
, MF=(M,list addr,COMPLETE)
, MF=(M,list addr,NOCHECK)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default value.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms of IEFSJSYM in the following order:
1. Use IEFSJSYM ...MF=(M,list-addr,COMPLETE) to specify appropriate parameters, including all required parameters.
2. Use IEFSJSYM ...MF=(M,list-addr,NOCHECK) to specify the parameters that you want to change.
3. Use IEFSJSYM ...MF=(E,list-addr,NOCHECK) to execute the macro.

,list addr
Specifies the name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this can be an RS-type address or an address in register (2)-(12).
IEFSJSYM macro

\attr
Specifies an optional 1-60 character input string which forces boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

\COMPLETE
Specifies for the system to check for required parameters, and to supply default values for omitted optional parameters.

\NOCHECK
Specifies for the system to not check for required parameters, and to not supply default values for omitted optional parameters.

\NUMENTRIES=numentries
A required input parameter for REQUEST=GETBYNAME that specifies the number of entries in the CHAR(16) array pointed to by SYMLISTARRAY. To code this parameter, specify an RS-type address or address in register (2)-(12) of a halfword field, or specify a literal decimal value.

\PLISTVER=IMPLIED_VERSION
\PLISTVER=MAX
\PLISTVER=0
Specifies the version of the macro and determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using this parameter, specify it on all macro forms used for a request and with the same value on all of the macro forms. To code this parameter, specify IMPLIED VERSION, MAX, or 0, as follows:

IMPLIED VERSION
Specifies the lowest version that allows all of the specified parameters to be processed. If you omit the PLISTVER parameter, IMPLIED VERSION is the default.

MAX
Specifies to maximize the parameter list size. Because the supported maximum size can grow from release to release, the amount of storage that your program requires can also change. If your system can tolerate a size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all of the parameters that you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

0
Specifies to use the currently available parameters.

REQUEST=GETALL
REQUEST=GETBYNAME
A required parameter that specifies the JCL symbols to get. Use REQUEST=GETALL to get all symbol values that were exported. Use REQUEST=GETBYNAME to get specific named symbol values, given an array of symbol names.

RETCODE=retcode
An optional output parameter into which the return code is copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without
IEFSJSYM macro

parentheses), the value will be left in GPR 15. To code this parameter, specify an RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

\[RSNDCODE=rsncode\]

An optional output parameter into which the reason code is copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0. To code this parameter, specify an RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), (REG0), (REG00), or (R0).

\[SYMBAREALEN=symbarealen\]

A required input parameter that specifies the length of the SYMBAREA that is provided by the caller. To code this parameter, specify an RS-type address, or address in register (2)-(12) of a fullword field, or specify a literal decimal value.

\[SYMLISTARRAY=symlistarray\]

A required input parameter for \texttt{REQUEST=GETBYNAME} that contains an array of up to 16 character entries, each of which contains a symbol name for which the symbol value is to be returned. Symbol names must be left-justified in the array entry, and if shorter than 16 characters, padded on the right with blank spaces.

A wildcard character (asterisk (*) to match 0 or more characters in the symbol name, or question mark (?) to match exactly one character) can used to specify a generic symbol name. Symbol names should not contain a leading ampersand character (&) or any special character other than a wildcard character.

IEFSJSYM returns a null value (\texttt{SYDESYMVALUELEN=0}) for entries that do not contain valid JCL symbol name; in addition, a return code of \texttt{IEFSJSYMRC_Warn} and a reason code of \texttt{IEFSJSYM\_SymbolNameNotProcessed} are set. To code this parameter, specify an RS-type address, or address in register (2)-(12), of a character field.

\section*{ABEND codes}

None.

\section*{Return and reason codes}

Table 74 contains return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

\begin{table}
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{Return Code Decimal (hex)} & \textbf{Reason Code Decimal (hex)} & \textbf{Equate Symbol for Reason Code Meaning and Action} \\
\hline
00 (00) & None. & \textbf{Equate Symbol:} \texttt{IEFSJSYMRC\_Ok} \\
 & & \textbf{Meaning:} The requested function was successfully completed. \\
 & & \textbf{Action:} None. \\
\hline
\end{tabular}
\caption{Return and reason codes for the IEFSJSYM macro}
\end{table}
### Table 74. Return and reason codes for the `IEFSJSYM` macro (continued)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>04 (04)</td>
<td>None.</td>
<td>Equate Symbol: <code>IEFSJSYMRC_Warn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The values of some of the requested symbols could not be returned. Symbols without values are returned with null values and symbol value lengths of 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Refer to the action for the individual reason code.</td>
</tr>
<tr>
<td>04 (04)</td>
<td>04 (04)</td>
<td>Equate Symbol: <code>IEFSJSYMResn_SymbolNameNotProcessed</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> For a GETBYNAME request, the values of some of the requested symbols could not be returned. This can occur if the symbol was not exported by the calling JCL, if the symbol was not SET after being exported, or if the symbol name input to <code>IEFSJSYM</code> did not follow JCL symbol name conventions. This only occurs for specific symbol names in the array, and not to symbol names that include wildcard characters. Symbols without values are returned with null values and symbol value lengths of 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check the submitted JCL to ensure that an EXPORT was done for the requested symbol and that the symbol was SET after the EXPORT statement.</td>
</tr>
<tr>
<td>04 (04)</td>
<td>08 (008)</td>
<td>Equate Symbol: <code>IEFSJSYMResn_InSuffSymSpace</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> Insufficient space to return all of the symbols and values requested.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Use the value returned in SYDALEN to obtain the storage required to fit all of the returned symbols, values and control information.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>None.</td>
<td>Equate Symbol: <code>IEFSJSYMRC_ParmError</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> Invalid input parameter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Refer to the action for the individual reason code.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>08 (04)</td>
<td>Equate Symbol: <code>IEFSJSYMResn_ParmlistAddrInvalid</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> <code>IEFSJSYM</code> could not use the parameter list provided.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Verify that the address of the parameter list is valid and resides in virtual storage of the primary address space.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>08 (08)</td>
<td>Equate Symbol: <code>IEFSJSYMResn_SymbareaAddrInvalid</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> <code>IEFSJSYM</code> could not use the output symbol area provided.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Verify that the address of the symbol area is valid and resides in virtual storage of the primary address space.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>08 (00C)</td>
<td>Equate Symbol: <code>IEFSJSYMResn_SymbListAddrInvalid</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> <code>IEFSJSYM</code> could not use the input symbol list provided.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Verify that the address of the data area is valid and resides in virtual storage of the primary address space.</td>
</tr>
<tr>
<td>08 08</td>
<td>08 (010)</td>
<td>Equate Symbol: <code>IEFSJSYMResn_Mismatched_VersLen</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The length of the <code>IEFSJSYM</code> parameter list does not match the version number supplied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Ensure that the parameter list that was built matches the specified or default parameter list version.</td>
</tr>
</tbody>
</table>
### Table 74. Return and reason codes for the IFSJSYM macro (continued)

|--------------------------|---------------------------|-------------------------------|--------------------|
| 08 08                    | 08 (014)                  | **Equate Symbol:** IEFSJSYMRsn_Unsupported_version | **Meaning:** The version of the parameter list is not supported with this level of the IFSJSYM service.  
**Action:** Correct the version and other parameters to match the system where the job was run, or run the job on a system that supports this version of IFSJSYM. |
| 08 (08)                  | 08 (18)                   | **Equate Symbol:** IEFSJSYMRsn_Unsupported_Function | **Meaning:** The request was for a function that is not supported with this level of the IFSJSYM service.  
**Action:** Choose a function that is supported on this level of IFSJSYM service, or request this function on a system that supports it. |
| 0C (0C)                  | None                      | **Equate Symbol:** IEFSJSYMRsn_InsSuffHdSpace | **Meaning:** Insufficient space to return the header (SYDHDR) portion of the data area.  
**Action:** Refer to the mapping macro IFSJSYM and pass an area at least as large as the DSECT SYDHDR. |
| 0C (0C)                  | 0C (04)                   | **Equate Symbol:** IEFSJSYMRsn_StorageNotObtained | **Meaning:** Failed to obtain above the bar storage via IARV64. The length of the storage requested is based on the size of the caller's SYMBAREA and SYMLISTARRAY size.  
**Action:** Check the SYMBAREALEN specification. If SYMBAREALEN is coded as an extremely large number, try reducing the SYMBAREALEN to a size that is comparable to the number of symbols requested. |
| 0C (0C)                  | 0C (08)                   | **Equate Symbol:** IEFSJSYMRsn_IncorrectExecEnv | **Meaning:** A proper execution environment does not exist for the service.  
**Action:** Verify that the program meets the requirements described previously. The returned DIAGDATA value will contain information that identifies the problem. |
| 0C (0C)                  | 0C (010)                  | **Equate Symbol:** IEFSJSYMRsn_UnexpectedSjfResponse | **Meaning:** Underlying JCL services invoked by the service returned with an unexpected return and reason codes. IFSJSYM might have been invoked before the job execution environment was established, or after the job execution environment had ended.  
**Action:** Verify that the program is running under a batch program environment. The returned DIAGDATA value contains information, such as the JCL service and return and reason codes, to diagnose the error. |
| 10 (10)                  | None                      | **Equate Symbol:** IEFSJSYMRn_IntError | **Meaning:** Unexpected internal error.  
**Action:** Report the problem to the system programmer. |
IEFSJSYM macro

Example

SJSYM_RC  DS  F
SJSYM_RSN  DS  F
SJDIAG   DS  4F
SYMBOLS  DS  0D
S1       DC C'DSN '
S2       DC C'VOL '
S3       DC C'UNIT '
SYMBOLAREA DS  64F
    IEFSJSYM SYMLISTARRAY=SYMBOLS,NUMENTRIES=3,SYMBAREAL=SYMBOLAREA,
             SYMBAREALEN=256,DIAGDATA=SJDIAG
or
    IEFSJSYM REQUEST=GETALL,SYMBAREAL=SYMBOLAREA,SYMBAREALEN=256,
             RETCODE=SJSYM_RC,RSNCODE=SJSYM_RSN,DIAGDATA=SJDIAG
IEFSJSYM macro
Chapter 80. IEFSSI — Dynamically control a subsystem

Description

Use the IEFSSI macro to dynamically control a subsystem in any of the following ways:

- Adding and defining a subsystem to the system
- Activating a subsystem so that its function routines can process function requests
- Defining a set of optional subsystem characteristics
- Deactivating a subsystem
- Swapping the current SSVT with a new SSVT
- Storing subsystem-defined data for a subsystem
- Retrieving subsystem-defined data for a subsystem that was previously stored with the put request
- Query information for all subsystems defined to the SSI

The requests for the macro are:

- IEFSSI REQUEST=ADD, dynamically adds and defines a subsystem to the system. See "REQUEST=ADD parameter of IEFSSI" on page 825 for the syntax of this request.
- IEFSSI REQUEST=ACTIVATE, dynamically activates a subsystem so that its function routines are available to process function requests. See "REQUEST=ACTIVATE parameter of IEFSSI" on page 829 for the syntax of this request.
- IEFSSI REQUEST=OPTIONS, which defines a set of optional subsystem characteristics. See "REQUEST=OPTIONS parameter of IEFSSI" on page 832 for the syntax of this request.
- IEFSSI REQUEST=DEACTIVATE, which deactivates a subsystem. See "REQUEST=DEACTIVATE parameter of IEFSSI" on page 836 for the syntax of this request.
- IEFSSI REQUEST=SWAP, which replaces the SSVT that is currently being used to route function requests with a new one. See "REQUEST=SWAP parameter of IEFSSI" on page 839 for the syntax of this request.
- IEFSSI REQUEST=PUT, which stores subsystem-defined data for the subsystem. See "REQUEST=PUT parameter of IEFSSI" on page 843 for the syntax of this request.
- IEFSSI REQUEST=GET, which retrieves subsystem-defined data previously stored using the IEFSSI REQUEST=PUT service. See "REQUEST=GET parameter of IEFSSI" on page 846 for the syntax of this request.
- IEFSSI REQUEST=QUERY, which obtains information about a currently defined subsystem. See "REQUEST=QUERY parameter of IEFSSI" on page 849 for the syntax of this request.

The IEFSSI macro (REQUEST=QUERY only) is also described in z/OS MVS Programming: Assembler Services Reference ABE-HSI.

For ease of use, the standard form of the macro is shown for each IEFSSI request. The requests are described on the following pages along with the:
IEFSSI macro

- Standard form syntax diagram
- Description of the parameters

The following information is described once at the beginning of the IEFSSI macro description:
- Environment
- Programming requirements
- Restrictions
- Input register information
- Output register information
- Performance implications

Following the descriptions of the standard forms of all requests are:
- Abend codes
- Return and reason codes
- Examples

The REQUEST=ADD, REQUEST=ACTIVATE, REQUEST=OPTIONS, REQUEST=DEACTIVATE, REQUEST=SWAP, REQUEST=PUT, REQUEST=GET and REQUEST=QUERY parameters, which designate the services of the IEFSSI macro, are mutually exclusive. You can select only one.

For information about using dynamic subsystem services, see [z/OS MVS Using the Subsystem Interface](https://www.ibm.com). This topic also includes information about related macros IEFSSVT and IEFSSVTI.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>For the QUERY request, problem state with any PSW key.</td>
</tr>
<tr>
<td></td>
<td>The REQUEST=ADD, REQUEST=ACTIVATE,</td>
</tr>
<tr>
<td></td>
<td>REQUEST=OPTIONS, REQUEST=DEACTIVATE,</td>
</tr>
<tr>
<td></td>
<td>REQUEST=SWAP, REQUEST=PUT, and REQUEST=GET parameters require one of the following:</td>
</tr>
<tr>
<td></td>
<td>- Supervisor state</td>
</tr>
<tr>
<td></td>
<td>- Any system PSW key</td>
</tr>
<tr>
<td></td>
<td>- PSW key mask (PKM) allowing key 0-7</td>
</tr>
<tr>
<td></td>
<td>- APF authorization</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24-bit or 31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or Access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space</td>
</tr>
</tbody>
</table>

Programming requirements

- Include the CVT and IEFJESCT mapping macros in your program.
- Include the IEFJSRC mapping macro in your program. This macro defines the dynamic SSI return and reason codes.
IEFSSI macro

- For the REQUEST=QUERY parameter, the caller must include the IEFJSQRY macro to map the REQUEST=QUERY output.
- For the REQUEST=ACTIVATE and REQUEST=SWAP parameters, the subsystem must have created at least one SSI-managed vector table. An SSI-managed vector table is a vector table created with the IEFSSVT REQUEST=CREATE macro.

Restrictions

The caller must not have established an EUT FRR.

Input register information

Before issuing the IEFSSI macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2 - 13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2 - 13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14 - 15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

REQUEST=ADD parameter of IEFSSI

The IEFSSI macro with the ADD parameter dynamically adds and defines a subsystem to the system.
Syntax for REQUEST=ADD

The syntax of the IEFSSI REQUEST=ADD macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSI.</td>
</tr>
<tr>
<td>IEFSSI</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFSSI.</td>
</tr>
<tr>
<td>SUBNAME=subname</td>
<td>subname: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,REQUEST=ADD</td>
<td></td>
</tr>
<tr>
<td>,CONSNAME=consname</td>
<td>consname: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,CONSNAME=0</td>
<td><strong>Default</strong>: CONSNAME=0</td>
</tr>
<tr>
<td>,INITRTN=initrtn</td>
<td>initrtn: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,INITRTN=NO_INITRTN</td>
<td><strong>Default</strong>: INITRTN=NO_INITRTN</td>
</tr>
<tr>
<td>,INITPARM=initparm</td>
<td>initparm: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,INITPARM=NO_INITPARM</td>
<td><strong>Default</strong>: INITPARM=NO_INITPARM</td>
</tr>
<tr>
<td>,INITPLEN=initplen</td>
<td>initplen: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td><strong>Default</strong>: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=plistver</td>
<td>plistver: 1</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or address in register (2) - (12) of fullword output variable</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or address in register (2) - (12) of fullword output variable</td>
</tr>
<tr>
<td>,COM=com</td>
<td>com: comment string</td>
</tr>
<tr>
<td>,COM=NULL</td>
<td><strong>Default</strong>: COM=NULL</td>
</tr>
<tr>
<td>,MF=S</td>
<td><strong>Default</strong>: MF=S</td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td></td>
</tr>
</tbody>
</table>
### Parameters for REQUEST=ADD

The parameters are explained as follows:

**SUBNAME=**<subname>

A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name.

This fullword field must be padded to the right with blanks or nulls if it is less than 4 characters long.

When selecting subsystem names, note the following:

- If you specify a subsystem name with the characters '*' and '?', the DISPLAY SSI command or the IEFSSI REQUEST=QUERY service specifying that subsystem name may return information about subsystems other than this one. The '*' and '?' are treated as wildcard characters for these services.
- If you specify a subsystem name of '!PRI', the DISPLAY SSI command or the IEFSSI REQUEST=QUERY service specifying that subsystem name returns information about the primary subsystem, even though there is a subsystem named '!PRI'.

**Note:** If you need to start the subsystem, its name must meet the requirements for the name of a started task. See [z/OS MVS JCL Reference](#) for more information.

**REQUEST=ADD**

A parameter that specifies that a subsystem is to be dynamically defined.

**CONSNAME=**<consname>

An optional 8-character parameter that specifies the name (or an address in a register) of a console to which any messages the SSI issues as part of the initialization process are written. If an INITRTN parameter is specified, the console name is passed to the routine named on INITRTN.

The default is 0. If the default parameter is used, the SSI issues messages to the consoles that are receiving the INTIDS attribute.

**INITRTN=**<initrtn>

An optional 8-character parameter that specifies the name (or an address in a register) of a subsystem initialization routine.

A subsystem initialization routine name that is less than 8 characters long must be padded to the right with blanks. The default is NO_INITRTN.

**INITPARM=**<initparm>

An optional parameter that specifies the name (or address in a register) of a parameter string that is passed to the subsystem initialization routine. This...
IEFSSI macro

A parameter string can be up to 60 characters long. The INITPLEN parameter specifies the actual length of the passed parameter.

The INITPARM parameter is applicable only if you specify the INITRTN parameter.

```
,INITPLEN=initplen
```

A required parameter that contains the length of the parameter string to be passed to the subsystem initialization routine. You must specify this 4-byte parameter if you specify the INITPARM parameter.

INITPLEN can be from 1 to 60 characters long inclusive. If the length is greater than 60, the subsystem is defined but the subsystem initialization routine is not invoked.

```
,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=plistver
```

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

**IMPLIED_VERSION**

The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

**MAX**

The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1 The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

```
,RETCODE=retcode
```

An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

```
,RSNCODE=rsnocode
```

An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

```
,COM=com
,COM=NULL
```

An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about
the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

MF=S
MF=(L,list addr)
MF=(L,list addr,attr)
MF=(L,list addr,0D)
MF=(E,list addr)
MF=(E,list addr,COMPLETE)

Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

list addr
A required parameter that specifies the name of a storage area for the parameter list.

attr
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=ACTIVATE parameter of IEFSSI
The IEFSSI macro with the ACTIVATE parameter dynamically activates a subsystem so that its function routines are available to process function requests.

Syntax for REQUEST=ACTIVATE
The syntax of the IEFSSI REQUEST=ACTIVATE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSI.</td>
</tr>
<tr>
<td>IEFSSI</td>
<td></td>
</tr>
</tbody>
</table>
IEFSSI macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFSSI.</td>
</tr>
<tr>
<td>SUBNAME={subname}</td>
<td>subname: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,REQUEST=ACTIVATE</td>
<td></td>
</tr>
<tr>
<td>,INTOKEN={intoken}</td>
<td>intoken: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,INTOKEN=NO_INPUT_TOKEN</td>
<td>Default: INTOKEN=NO_INPUT_TOKEN</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER={plistver}</td>
<td>plistver: 1</td>
</tr>
<tr>
<td>,RETCODE={retcode}</td>
<td>retcode: RS-type address or address in register (2) - (12) of fullword output variable</td>
</tr>
<tr>
<td>,RNCODE={rsncode}</td>
<td>rsncode: RS-type address or address in register (2) - (12) of fullword output variable</td>
</tr>
<tr>
<td>,COM={com}</td>
<td>com: comment string</td>
</tr>
<tr>
<td>,COM=NULL</td>
<td>Default: COM=NULL</td>
</tr>
<tr>
<td>,MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

Parameters for REQUEST=ACTIVATE

The parameters are explained as follows:

| SUBNAME={subname}       | A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services. This field must be padded to the right with blanks or nulls if it is less than 4 characters long. |
| REQUEST=ACTIVATE       | A parameter that specifies a subsystem is to be dynamically activated so that its function routines are available to process function requests. Before invoking the IEFSSI macro and issuing the REQUEST=ACTIVATE parameter, the |
subsystem must be defined to the system, and you must ensure that an SSVT has been built using the IEFSSVT macro with the REQUEST=CREATE parameter.

The ACTIVATE request may also be used to re-activate a subsystem that has been deactivated. To re-activate a subsystem, you can either use the same SSVT as you used to deactivate the subsystem or you can use a new SSVT.

,INTOKEN=intoken
,INTOKEN=NO_INPUT_TOKEN
An optional 32-bit parameter that specifies the name (or an address in a register) of an input token that represents the SSVT that is used when activating the subsystem. The function routines associated with the SSVT are made available for processing requests.

The token must be one that was returned by one of the following:
• IEFSSVT REQUEST=CREATE
• IEFSSI REQUEST=DEACTIVATE
• IEFSSI REQUEST=SWAP

If the INTOKEN parameter is omitted, an SSVT is chosen as follows:
• The most recently active SSI-managed vector table
• The last SSI-managed vector table created, if no SSI-managed vector table has been activated

The default is NO_INPUT_TOKEN.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=plistver
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1 The currently available parameters.

To code: specify in this input parameter one of the following:
• IMPLIED_VERSION
• MAX
• A decimal value of 1
**IEFSSI macro**

\,Retcode=retcode
   An optional 4-byte parameter that specifies the name of an output field (or a
register) where the system places the return code. The return code is copied
from general purpose register 15.

\,RSNCODE=rsncode
   An optional 4-byte parameter that specifies the name of an output field (or a
register) where the system places the reason code. The reason code is copied
from general purpose register 0.

\,COM=com
\,COM=NULL
   An optional parameter that specifies the character input that appears in the
block comment before the macro invocation. Use it to make comments about
the macro invocation. The comment must be enclosed in quotation marks if it
contains any lower case characters. The default is NULL.

\,MF=S
\,MF=(L,list addr)
\,MF=(L,list addr,attr)
\,MF=(L,list addr,0D)
\,MF=(E,list addr)
\,MF=(E,list addr,COMPLETE)
   Use MF=S to specify the standard form of the IEFSSI macro, which builds an
in-line parameter list and generates the macro invocation to transfer control to
the service.

   Use MF=L to specify the list form of the IEFSSI macro. Use the list form
together with the execute form of the macro for applications that require
reentrant code. The list form defines an area of storage that the execute form
uses to store the parameters. No other parameters may be coded with the list
form of the macro.

   Use MF=E together with the list form of the macro for applications that require
reentrant code. The execute form of the IEFSSI macro stores the parameters
into the storage area defined by the list form and generates the macro
invocation to transfer control to the service.

\,list addr
   A required parameter that specifies the name of a storage area for the
parameter list.

\,attr
   An optional 1- to 60-character input string that contains any value that is
valid on an assembler DS pseudo-op. You can use this parameter to force
boundary alignment of the parameter list. If you do not code attr, the
system provides a value of 0D, which forces the parameter list to a
doubleword boundary.

\,COMPLETE
   An optional parameter that specifies that the system checks for required
parameters and supply defaults for omitted optional parameters. This is
the default parameter.

**REQUEST=OPTIONS parameter of IEFSSI**

The IEFSSI macro with the OPTIONS parameter defines a set of optional
subsystem characteristics.
### Syntax for REQUEST=OPTIONS

The syntax of the IEFSSI REQUEST=OPTIONS macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>One or more blanks must precede IEFSSI.</td>
</tr>
<tr>
<td>IEFSSI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>One or more blanks must follow IEFSSI.</td>
</tr>
<tr>
<td><code>SUBNAME=subname</code></td>
<td><em>subname</em>: RS-type address or address in register (2) - (12) of fullword output variable</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>REQUEST=OPTIONS</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>COM</strong>=NO  <em>Default: COMMAND=NO</em></td>
</tr>
<tr>
<td></td>
<td><strong>COM</strong>=YES  *RETD SUB=MSTR</td>
</tr>
<tr>
<td></td>
<td><strong>REQUEST=OPTIONS</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>COM</strong>=NULL  <em>Default: COM=NULL</em></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>MF</strong>=S  *Default: MF=S</td>
</tr>
<tr>
<td></td>
<td><strong>MF</strong>=(L,list addr)</td>
</tr>
<tr>
<td></td>
<td><strong>MF</strong>=(L,list addr,attr)</td>
</tr>
<tr>
<td></td>
<td><strong>MF</strong>=(L,list addr,0D)</td>
</tr>
<tr>
<td></td>
<td><strong>MF</strong>=(E,list addr)</td>
</tr>
<tr>
<td></td>
<td><strong>MF</strong>=(E,list addr,COMPLETE)</td>
</tr>
</tbody>
</table>
Parameters for REQUEST=OPTIONS

The parameters are explained as follows:

**SUBNAME=**subname

A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

**REQUEST=OPTIONS**

A parameter that specifies the definition of a set of optional subsystem characteristics. You can set the following subsystem options using this macro:

- Whether the subsystem responds to SETSSI commands
- Whether you want the invoking subsystem to start under the MSTR or primary subsystem

IEFSSI REQUEST=OPTIONS is the only way you can specify these optional characteristics.

If you invoke the OPTIONS parameter multiple times for a single subsystem, the most recent invocation determines the resulting characteristics. The defaults listed for the COMMAND and REQDSUB parameters below apply to the first invocation.

If a parameter is not specified on a subsequent invocation, the corresponding subsystem characteristic retains the value that was assigned by the last invocation that specified the parameter.

**COMMAND=**NO
**COMMAND=**YES

An optional parameter that specifies the whether the subsystem responds to the following commands:

- SETSSI ACTIVATE
- SETSSI DEACTIVATE

The meanings are:

**NO** The subsystem does not allow SETSSI commands. No is the default.

**YES** The subsystem allows SETSSI commands.

You need to specify COMMAND=YES only if the subsystem can tolerate the processing associated with each of the SETSSI commands listed above.

**REQDSUB=MSTR**
**REQDSUB=PRI**

An optional parameter that specifies whether a START subsystem command causes the subsystem to start under either the MSTR subsystem or the primary subsystem (JES).

When the procedure name on the START command matches a defined subsystem name, the procedure being started is recognized as a subsystem. If
IEFSSI macro

the START command does not specify the SUB parameter, the subsystem is started under the control of the subsystem identified by the REQDSUB parameter.

The meanings for REQDSUB=MSTR and REQDSUB=PRI are:

- **REQDSUB=MSTR** — The subsystem specified does not require the services of the primary subsystem and starts under the MSTR subsystem. MSTR is the default.
- **REQDSUB=PRI** — The subsystem specified requires the services of the primary subsystem and must start under its control. If a START subsystem command is issued before the primary subsystem is available, the processing that the subsystem was doing in response to the START command fails.

,**PLISTVER=IMPLIED_VERSION**
,**PLISTVER=MAX**
,**PLISTVER=plistver**

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

**IMPLIED_VERSION**

The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

**MAX**

The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

**1**

The currently available parameters.

**To code:** specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,**RETCODE=retcode**

An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,**RSNCODE=rsncode**

An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,**COM=com**
,**COM=NULL**

An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about
the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,list addr)
,MF=(L,list addr,attr)
,MF=(L,list addr,0D)
,MF=(E,list addr)
,MF=(E,list addr,COMPLETE)

Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

**REQUEST=DEACTIVATE parameter of IEFSSI**

The IEFSSI macro with the DEACTIVATE parameter deactivates a subsystem.

**Syntax for REQUEST=DEACTIVATE**

The syntax of the IEFSSI REQUEST=DEACTIVATE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSI.</td>
</tr>
<tr>
<td>IEFSSI</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFSSI.</td>
</tr>
</tbody>
</table>
### Parameters for REQUEST=DEACTIVATE

The parameters are explained as follows:

**SUBNAME=** subname  
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.  
This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

**REQUEST=** DEACTIVATE  
A parameter that specifies that a subsystem is to be deactivated. This stops any new function requests from being passed to the subsystem function routines.  
A subsystem can be reactivated after being deactivated by using the same or a different SSVT.
IEFSSI macro

,OUTTOKEN=outtoken
An optional 32-bit parameter that specifies the name (or an address in a register) of an output token. This is where the token that represents the deactivated SSVT is returned.

This token may be used in a subsequent ACTIVATE request to reactivate the subsystem using the same SSVT.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=plistver
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1 The currently available parameters.

To code: specify in this input parameter one of the following:
• IMPLIED_VERSION
• MAX
• A decimal value of 1

,RETCODE=retcode
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=rsncode
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=com
,COM=NULL
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,list addr)
,MF=(L,list addr,attr)
Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

A required parameter that specifies the name of a storage area for the parameter list.

An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

**REQUEST=SWAP parameter of IEFSSI**

The IEFSSI macro with the SWAP parameter replaces the SSVT that is currently being used to route function requests with a new one.

**Syntax for REQUEST=SWAP**

The syntax of the IEFSSI REQUEST=SWAP macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSI.</td>
</tr>
<tr>
<td>IEFSSI</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFSSI.</td>
</tr>
<tr>
<td>SUBNAME=substring</td>
<td>substring: RS-type address or address in register (2) - (12).</td>
</tr>
</tbody>
</table>
### Parameters for REQUEST=SWAP

The parameters are explained as follows:

**SUBNAME=subname**

A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

**REQUEST=SWAP**

A parameter that specifies the replacement of the SSVT currently being used to route function requests with a new SSVT. The current SSVT is deactivated and the new SSVT is (re)activated. The subsystem remains continuously active during this process.
When you use a SWAP request to switch SSVTs, it is possible for you to use a subsequent SWAP request to switch the SSVTs again, which restores the old function routines.

A SWAP request that targets an inactive subsystem is treated as an ACTIVATE request, but receives the IEFSSI_WARNING (4) return code.

**INTOKEN=intoken**

**INTOKEN=NO_INPUT_TOKEN**

An optional 32-bit parameter that specifies the name (or an address in a register) of an input token that represents the SSVT that is used when activating the subsystem. The function routines associated with the SSVT are made available for processing requests.

The token must be one that was returned by either the:

- IEFSSVT REQUEST=CREATE
- IEFSSI REQUEST=DEACTIVATE
- IEFSSI REQUEST=SWAP

If the INTOKEN parameter is omitted, an SSVT is chosen as follows:

- The most recently active SSI-managed vector table
- The last SSI-managed vector table created, if no SSI-managed vector table has been activated

The default is NO_INPUT_TOKEN.

**OUTTOKEN=outtoken**

An optional 32-bit parameter that specifies the name (or an address in a register) of an output token. This is where the token that represents the deactivated SSVT is returned.

This token may be used in a subsequent SWAP request to reactivate the subsystem using the same SSVT.

**PLISTVER=IMPLIED_VERSION**

**PLISTVER=MAX**

**PLISTVER=plistver**

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

**IMPLIED_VERSION**

The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

**MAX**

The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1 The currently available parameters.
**IEFSSI macro**

**To code:** specify in this input parameter one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 1

`RETCODE=retcode`
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

`RSNCODE=rsncode`
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

`COM=com`
`COM=NULL`
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

`MF=S`
`MF=(L,list addr)`
`MF=(L,list addr,attr)`
`MF=(L,list addr,0D)`
`MF=(E,list addr)`
`MF=(E,list addr,COMPLETE)`
Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

`list addr`
A required parameter that specifies the name of a storage area for the parameter list.

`attr`
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

`COMPLETE`
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.
**REQUEST=PUT parameter of IEFSSI**

The IEFSSI macro with the PUT parameter stores subsystem-defined data for the subsystem.

**Syntax for REQUEST=PUT**

The syntax of the IEFSSI REQUEST=PUT macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>SUBNAME=subname</td>
<td><em>subname</em>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,REQUEST=PUT</td>
<td></td>
</tr>
<tr>
<td>,SUBDATA1=subdata1</td>
<td><em>subdata1</em>: RS-type address or address in register (2) - (12). of a 4-character input area</td>
</tr>
<tr>
<td>,SUBDATA2=subdata2</td>
<td><em>subdata2</em>: RS-type address or address in register (2) - (12). of a 4-character input area</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td><strong>Default</strong>: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=plistver</td>
<td><em>plistver</em>: 1</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td><em>retcode</em>: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td><em>rsncode</em>: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,COM=com</td>
<td><em>com</em>: comment string</td>
</tr>
<tr>
<td>,COM=NULL</td>
<td><strong>Default</strong>: COM=NULL</td>
</tr>
<tr>
<td>,MF=S</td>
<td><strong>Default</strong>: MF=S</td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L.list addr.attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td></td>
</tr>
</tbody>
</table>
IEFSSI macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters for REQUEST=PUT**

The parameters are explained as follows:

**SUBNAME=**subname

A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services. This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

**REQUEST=**PUT

A parameter that specifies the storage of subsystem-defined data for the subsystem. Two non-contiguous 4-byte fields are available for the subsystem data. One of these fields is typically used to anchor subsystem specific control blocks.

You must specify at least one of the following parameters:

**SUBDATA1=**subdata1

The name (or address in a register) of a 4-character input area that holds the first 4-bytes of subsystem specific information.

**SUBDATA2=**subdata2

The name (or address in a register) of a 4-character input area that holds the second 4-bytes of subsystem specific information.

**PLISTVER=**IMPLIED_VERSION

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

**IMPLIED_VERSION**

The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

**MAX**

The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1 The currently available parameters.

**To code:** specify in this input parameter one of the following:
IEFSSI macro

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=retcode
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=rsncode
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=com
,COM=NULL
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,list addr)
,MF=(L,list addr,attr)
,MF=(L,list addr,0D)
,MF=(E,list addr)
,MF=(E,list addr,COMPLETE)
Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.
### IEFSSI macro

#### REQUEST=GET parameter of IEFSSI

The IEFSSI macro with the GET parameter retrieves subsystem-defined data previously stored using the IEFSSI REQUEST=PUT request.

#### Syntax for REQUEST=GET

The syntax of the IEFSSI REQUEST=GET macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSI.</td>
</tr>
<tr>
<td>IEFSSI</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFSSI.</td>
</tr>
<tr>
<td>SUBNAME=subname</td>
<td><em>subname</em>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,REQUEST=GET</td>
<td></td>
</tr>
<tr>
<td>,SUBDATA1=subdata1</td>
<td><em>subdata1</em>: RS-type address or address in register (2) - (12) of a 4-character output area.</td>
</tr>
<tr>
<td>,SUBDATA2=subdata2</td>
<td><em>subdata2</em>: RS-type address or address in register (2) - (12) of a 4-character output area.</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td><strong>Default</strong>: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=plistver</td>
<td><em>plistver</em>: 1</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td><em>retcode</em>: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,RSNCODE=rsnocode</td>
<td><em>rsnocode</em>: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,COM=com</td>
<td><em>com</em>: comment string</td>
</tr>
<tr>
<td>,COM=NULL</td>
<td><strong>Default</strong>: COM=NULL</td>
</tr>
<tr>
<td>,MF=S</td>
<td><strong>Default</strong>: MF=S</td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td></td>
</tr>
</tbody>
</table>
Parameters for REQUEST=GET

The parameters are explained as follows:

**SUBNAME=subname**
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

**REQUEST=GET**
A parameter that specifies the retrieval of subsystem-defined data previously stored using the IEFSSI REQUEST=PUT request. Two non-contiguous 4-byte fields are available for the subsystem data.

You must specify at least one of the following parameters:

**SUBDATA1=subdata1**
The name (or address in a register) of a 4-character output area that holds the first 4-bytes of subsystem specific information identified by the SUBDATA1 parameter on a previous IEFSSI REQUEST=PUT request.

**SUBDATA2=subdata2**
The name (or address in a register) of a 4-character output area that holds the second 4-bytes of subsystem specific information identified by the SUBDATA2 parameter on a previous IEFSSI REQUEST=PUT request.

**PLISTVER=IMPLIED_VERSION**
**PLISTVER=MAX**
**PLISTVER=plistver**
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

**IMPLIED_VERSION**
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

**MAX**
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.
IEFSSI macro

To code: specify in this input parameter one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=retcode
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=rsncode
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=com,COM=NULL
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,list addr)
,MF=(L,list addr,attr)
,MF=(L,list addr,0D)
,MF=(E,list addr)
,MF=(E,list addr,COMPLETE)
Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.
**REQUEST=QUERY parameter of IEFSSI**

The IEFSSI macro with the QUERY parameter requests information about subsystems defined to the system.

**Syntax for REQUEST=QUERY**

The syntax of the IEFSSI REQUEST=QUERY macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede IEFSSI.</td>
</tr>
<tr>
<td>IEFSSI</td>
<td></td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must follow IEFSSI.</td>
</tr>
<tr>
<td><code>SUBNAME=subname</code></td>
<td><code>subname</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><code>,REQUEST=QUERY</code></td>
<td></td>
</tr>
<tr>
<td><code>,WORKAREA=workarea</code></td>
<td><code>workarea</code>: RS-type address or address in register (2) - (12) of an output area.</td>
</tr>
<tr>
<td><code>,WORKASP=workasp</code></td>
<td><code>workasp</code>: RS-type address or address in register (2) - (12) of an input area.</td>
</tr>
<tr>
<td><code>,PLISTVER=IMPLIED_VERSION</code></td>
<td></td>
</tr>
<tr>
<td><code>,PLISTVER=MAX</code></td>
<td><strong>Default</strong>: IMPLIED_VERSION</td>
</tr>
<tr>
<td><code>,PLISTVER=listver</code></td>
<td><code>listver</code>: 1</td>
</tr>
<tr>
<td><code>,RETCODE=retcode</code></td>
<td><code>retcode</code>: RS-type address or address in register (2) - (12). of fullword output variable</td>
</tr>
<tr>
<td><code>,RSNCODE=rsncode</code></td>
<td><code>rsncode</code>: RS-type address or address in register (2) - (12). of fullword output variable</td>
</tr>
<tr>
<td><code>,COM=com</code></td>
<td><code>com</code>: comment string</td>
</tr>
<tr>
<td><code>,COM=NULL</code></td>
<td><strong>Default</strong>: COM=NULL</td>
</tr>
<tr>
<td><code>,MF=S</code></td>
<td><strong>Default</strong>: MF=S</td>
</tr>
<tr>
<td><code>,MF=(L,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>,MF=(L,list addr,attr)</code></td>
<td></td>
</tr>
<tr>
<td><code>,MF=(L,list addr,0D)</code></td>
<td></td>
</tr>
<tr>
<td><code>,MF=(E,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>,MF=(E,list addr,COMPLETE)</code></td>
<td></td>
</tr>
</tbody>
</table>
Parameters for REQUEST=QUERY

The parameters are explained as follows:

**SUBNAME=**subname

A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

For the REQUEST=QUERY parameter, the subsystem name may contain the wildcard characters '*' and '?' to request information about multiple subsystems. The meanings for the wildcard characters are:

* Matches 0 or more characters.
  
  Use a SUBNAME parameter value of '*' to indicate that information is to be returned for all subsystems.

? Matches exactly 1 character.

  Use a SUBNAME parameter value of 'PRI' to indicate that information is to be returned for the primary subsystem.

**REQUEST=**QUERY

A parameter that specifies the request to obtain information about a currently defined subsystem named in the SUBNAME parameter.

The output from IEFSSI REQUEST=QUERY is mapped by the IEFJSQRY macro. Subsystems are listed in broadcast order, that is, the order in which they receive broadcast SSI requests.

**WORKAREA=**workarea

A required parameter that specifies a name (or register containing the address) of a pointer output field that contains the address of the subsystem information returned by the QUERY request.

The output area is mapped by the IEFJSQRY macro. The JQRYLEN field contains the length of the output area.

**WORKASP=**workasp

An optional parameter that specifies a name (or register containing the address) of a one-byte input field that specifies the subpool that the SSI uses to obtain a work area for the returned subsystem information. The caller is responsible for freeing this work area.

IBM recommends that you use a job-related or task-related subpool. This allows the system to free the associated storage when the job or task ends, if the caller does not free the returned area.

If WORKASP is not specified, the caller's subpool zero is used. Storage for the query information is obtained above 16 megabytes. AMODE 24 callers must switch into AMODE 31 to address this storage. Unauthorized callers may request storage only in the following unauthorized subpools:

- 0-127
- 131
IEFSSI macro

• 132

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=plistver

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION

The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX

The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1 The currently available parameters.

To code: specify in this input parameter one of the following:

• IMPLIED_VERSION
• MAX
• A decimal value of 1

,RETCODE=retcode

An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=rsncode

An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=com
,COM=NULL

An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,list addr)
,MF=(L,list addr,attr)
,MF=(L,list addr,0D)
,MF=(E,list addr)
,MF=(E,list addr,COMPLETE)

Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.
IEFSSI macro

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

\texttt{\textbackslash list addr}

A required parameter that specifies the name of a storage area for the parameter list.

\texttt{\textbackslash attr}

An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code \texttt{attr}, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

\texttt{\textbackslash COMPLETE}

An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

ABEND codes

An invocation of the IEFSSI macro may result in an abend code X'8C5'. See \textit{z/OS MVS System Codes} for an explanation of this abend code.

Return and reason codes

When the IEFSSI macro returns control to your program, GPR 15 (and \textit{retnode}, if you coded RETCODE) contains a return code. When the value in GPR 15 is not 0, GPR 0 (and \textit{rsncode} if you coded RSNCODE) contains the reason code.

The IEFJSRC mapping macro provides equate symbols for the return and reason codes. The equate symbols associated with each Return Code are:

<table>
<thead>
<tr>
<th>Decimal (Hex)</th>
<th>Equate Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>IEFSSI_SUCCESS</td>
</tr>
<tr>
<td>04 (04)</td>
<td>IEFSSI_WARNING</td>
</tr>
<tr>
<td>08 (08)</td>
<td>IEFSSI_INVALID_PARAMETERS</td>
</tr>
<tr>
<td>12 (0C)</td>
<td>IEFSSI_REQUEST_FAIL</td>
</tr>
<tr>
<td>20 (14)</td>
<td>IEFSSI_SYSTEM_ERROR</td>
</tr>
<tr>
<td>24 (18)</td>
<td>IEFSSI_UNAVAILABLE</td>
</tr>
</tbody>
</table>

The following table contains return and reason codes, the equate symbols associated with each reason code and the meaning and suggested action for each return and reason code.
### Table 75. Return and Reason Codes for the IEFSSI Macro

<table>
<thead>
<tr>
<th>Return Code decimal (hex)</th>
<th>Reason Code decimal (hex)</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00 (00)                   | 00 (00)                   | **Equate Symbol**: IEFSSI_FUNCTIONS_COMPLETE  
**Meaning**: The request completed successfully. The result depends on the request:  
• ADD — A subsystem has been added to the system  
• ACTIVATE — A subsystem has been activated  
• OPTIONS — A set of optional subsystem characteristics has been defined  
• DEACTIVATE — A subsystem has been deactivated  
• SWAP — The current SSVT has been swapped with a new SSVT  
• PUT — Subsystem-defined data has been stored  
• GET — Subsystem-defined data has been retrieved  
• QUERY — Information for all subsystems defined to the SSI has been queried  
**Action**: None. |
| 04 (04)                   | 300 (12C)                 | **Equate Symbol**: IEFSSI_DEACT_INACTIVE  
**Meaning**: The subsystem was already inactive. This is a DEACTIVATE request error.  
**Action**: None. |
| 04 (04)                   | 301 (12D)                 | **Equate Symbol**: IEFSSI_DEACT_OUT_VT_NOT_SSI  
**Meaning**: The subsystem is deactivated, however a previously active vector table was not SSI-managed. OUTTOKEN value is 0. This is a DEACTIVATE request error.  
**Action**: None. |
| 04 (04)                   | 500 (1F4)                 | **Equate Symbol**: IEFSSI_SWAP_INACTIVE  
**Meaning**: The subsystem was initially active. OUTTOKEN value is 0. This is a SWAP request error.  
**Action**: None. |
| 04 (04)                   | 501 (1F5)                 | **Equate Symbol**: IEFSSI_SWAP_OUT_VT_NOT_SSI  
**Meaning**: The swap is complete, however the previously active vector table was not SSI-managed. OUTTOKEN value is 0. This is a SWAP request error.  
**Action**: None. |
| 04 (04)                   | 900 (384)                 | **Equate Symbol**: IEFSSI_QUERY_INCOMPLETE  
**Meaning**: The data returned by the QUERY request may be incomplete. This is a QUERY request error.  
**Action**: Check the JQRY_INCOMPLETE flag for each subsystem that was queried. |
| 08 (08)                   | 00 (000)                  | **Equate Symbol**: IEFSSI_SUBSYSTEM_UNKNOWN  
**Meaning**: The subsystem is not defined to the SSI.  
**Action**: Correct the subsystem name or define a subsystem with either the IEFSSI macro or the SETSSI command. |
### Table 75. Return and Reason Codes for the IEFSSI Macro (continued)

<table>
<thead>
<tr>
<th>Return Code decimal (hex)</th>
<th>Reason Code decimal (hex)</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08 (08)                  | 04 (004)                  | **Equate Symbol**: IEFSSI_NON_DYNAMIC  
**Meaning**: The subsystem is not dynamic.  
**Action**: ReIPL the system and define the target subsystem with either the IEFSSI macro, the SETSSI command, or the keyword format IEFSSNxx parmlib member entry. Note that once a subsystem has been defined, it cannot be deleted or defined again as dynamic. |
| 08 (08)                  | 08 (008)                  | **Equate Symbol**: IEFSSI_BAD_VT_TOKEN  
**Meaning**: The SSVT token does not correspond to a valid SSVT table.  
**Action**: Correct the token. The token must be one that was returned by either the IEFSSVT REQUEST=CREATE macro, the IEFSSI REQUEST=DEACTIVATE macro, or the IEFSSI REQUEST=SWAP macro. |
| 08 (08)                  | 12 (00C)                  | **Equate Symbol**: IEFSSI_INVALID_NAME  
**Meaning**: The subsystem name or the routine name contains characters that are not valid.  
**Action**: Correct the subsystem name by removing the characters that are not valid. |
| 08 (08)                  | 16 (010)                  | **Equate Symbol**: IEFSSVT_INIT_PARMS  
**Meaning**: The initialization routine parameter string is too long.  
**Action**: Correct the parameter string so that it is no longer than 60 characters. |
| 12 (0C)                  | 100 (064)                 | **Equate Symbol**: IEFSSI_DUPLICATE_SUBSYSTEM  
**Meaning**: The subsystem already exists. This is an ADD request error.  
**Action**: Do not perform the ADD request if the existing subsystem is one you want. If the existing subsystem is not the one you want, select another name for the new subsystem, which does not conflict with the name of any existing subsystem name.  
You can use the IEFSSI REQUEST=QUERY macro to find all existing names. |
| 12 (0C)                  | 101 (065)                 | **Equate Symbol**: IEFSSI_INITRTN_NOT_FOUND  
**Meaning**: A usable copy of this initialization routine could not be found. This is an ADD request error. For example:  
- The module was not found.  
- The module was found, but was not APF-authorized.  
**Action**: Correct the initialization routine name or make sure it is present in either LINKLIB or LPALIB, and is APF authorized. |
| 12 (0C)                  | 102 (066)                 | **Equate Symbol**: IEFSSI_INITRTN_ABEND  
**Meaning**: The initialization routine ended abnormally. This is an ADD request error.  
**Action**: Check the dump produced by the abend and correct the problem with the initialization routine. |
### Table 75. Return and Reason Codes for the IEFSSI Macro (continued)

<table>
<thead>
<tr>
<th>Return Code decimal (hex)</th>
<th>Reason Code decimal (hex)</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 12 (0C)                   | 103 (0x7)                 | **Equate Symbol**: IEFSSI_ADD_STORAGE  
**Meaning**: Unable to obtain storage for the subsystem definition. This is an ADD request error.  
**Action**: Check the current use of the system storage to determine why storage was not available. Retry the request later in case storage has become available. |
| 12 (0C)                   | 200 (0xC8)                | **Equate Symbol**: IEFSSI_SUBSYSTEM_ACTIVE  
**Meaning**: The subsystem is already active. This is an ACTIVATE request error.  
**Action**: None. |
| 12 (0C)                   | 201 (0xC9)                | **Equate Symbol**: IEFSSI_ACT_NO_ELIGIBLE_VT  
**Meaning**: The SSVT is not specified and a valid default is not available. This is an ACTIVATE request error.  
**Action**: Provide an SSI-managed SSVT using the IEFSSVT REQUEST=CREATE macro. |
| 12 (0C)                   | 500 (1F4)                 | **Equate Symbol**: IEFSSI_SWAP_NO_ELIGIBLE_VT  
**Meaning**: The SSVT is not specified and a valid default is not available. This is a SWAP request error.  
**Action**: Provide an SSI-managed SSVT using the IEFSSVT REQUEST=CREATE macro. |
| 12 (0C)                   | 502 (1F6)                 | **Equate Symbol**: IEFSSI_SWAP_ALREADY_ACTIVE  
**Meaning**: The SSVT that is to be made active (specified by the INTOKEN field) is already active. This is a SWAP request error.  
**Action**: None. |
| 12 (0C)                   | 900 (384)                 | **Equate Symbol**: IEFSSI_QUERY_STORAGE  
**Meaning**: Unable to obtain storage for an output of the QUERY request. This a QUERY request error.  
**Action**: Check the current use of the system storage to determine why storage was not available. Retry the request later in case storage has become available. |
| 20 (14)                   | —                         | **Equate Symbol**: IEFSSI_SYSTEM_ERROR  
**Meaning**: System error  
**Action**: Investigate the following possible causes:  
• Inability to obtain a system resource  
• Abnormal task termination  
Obtain the system dump, if any, and contact the IBM support center. |
| 24 (18)                   | —                         | **Equate Symbol**: IEFSSI_UNAVAILABLE  
**Meaning**: The IEFSSI macro has been invoked too early during system initialization.  
**Action**: Delay the invocation of the IEFSSI macro to a later point in the IPL.
Example 1
Submit a request to add subsystem FRED, call the initialization routine and route all initialization messages to the FREDCONS console.

```
IEFSSI REQUEST=ADD,SUBNAME=SNAME,INITRTN=INITPGM, X
  INITPARM=IPARMS,INITPLEN=5,CONSNAME=ICONSOLE, X
  RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

```
SNAME DC CL4'FRED'
INITPGM DC CL8'FREDPGM'
IPARMS DC CL60'HELLO'
ICONSOLE DC CL8'FREDCONS'
```

```
WORKAREA DSECT OF
RETURN_CODE DS F
REASON_CODE DS F
WORKLEN EQU *-WORKAREA
```

Example 2
Activate subsystem FRED using the SSVT identified by the SSVTOK token. Assume that the SSVTOK token was returned by a previous invocation of the IEFSSVT REQUEST=CREATE macro.

```
IEFSSI REQUEST=ACTIVATE,SUBNAME=SNAME,INTOKEN=SSVTTOK, X
  RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

Example 3
Inform the system that the subsystem responds to SETSSI commands and requires the services of the primary subsystem when starting.

```
IEFSSI REQUEST=OPTIONS,SUBNAME=SNAME,COMMAND=YES,REQDSUB=PRI, X
  RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

Example 4
Deactivate a subsystem and return the token of the outgoing SSVT.
```
IEFSSI REQUEST=DEACTIVATE,SUBNAME=SNAME,OUTTOKEN=SSVTTOK, X
  RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

Example 5
Replace the current set of function routines being used by the subsystem with a new set of function routines. NEWTOK is a token previously returned by the IEFSSVT REQUEST=CREATE service. NEWTOK identifies the incoming SSVT.
```
IEFSSI REQUEST=SWAP,SUBNAME=SNAME,INTOKEN=NEWTOK, X
  OUTTOKEN=OLDTOK, X
  RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

Example 6
Store the address of the FREDCB subsystem control block for later retrieval by the subsystem function routines.
```
LA 5,FREDCB Get address of subsystem control block
ST 5,DATA1 Store address
LA 4,DATA1 Get address of field containing pointer
IEFSSI REQUEST=PUT,SUBNAME=SNAME,SUBDATA1=(4), X
  RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```
IEFSSI macro

Note: When using the register notation (4), the register contains the address of the data to be stored, not the data itself. The data stored in this case is the address of the FREDCB control block.

Example 7
Retrieve subsystem-defined data that was previously stored using the IEFSSI REQUEST=PUT service and place the retrieved data at the location whose address is contained in register 5.

```
IEFSSI REQUEST=GET, SUBNAME=SNAME, SUBDATA1=(5), X
  RETCODE=RETURN_CODE, RSNCODE=REASON_CODE
```

Example 8
Obtain subsystem information for all subsystems whose name begins with 'JES' and free the storage obtained by the SSI.

```
IEFSSI REQUEST=QUERY, SUBNAME=SNAME, X
  WORKAREA=WAREA, X
  RETCODE=RETURN_CODE, RSNCODE=REASON_CODE

: L R5, WAREA
  USING JQRY_HEADER, R5
L R8, JQRYLEN
STORAGE RELEASE, LENGTH=(0), ADDR=(R5)
:
SNAME DC CL4 'JES*'
WAREA DS A
IEFJSQRY
```
IEFSSI macro
Chapter 81. IEFSSVT — Create a subsystem vector table

Description

Use the IEFSSVT macro to:

- Create subsystem vector tables (SSVTs).
- Modify the subsystem response to function requests by:
  - Disabling existing function codes
  - Enabling new function codes
  - Exchange function routines for a supported function code

The IEFSSVT macro allows users to specify function routines by address or name rather than requiring the subsystem interface (SSI) to load the routines. This is useful if the subsystem wants to load its function routines into global storage, but does not want the routines to be deleted if the address space ends. In this case, the subsystem can perform a load-to-address, rather than a standard load, and pass the addresses to the IEFSSVT macro.

The requests for the macro are:

- IEFSSVT REQUEST=CREATE, which creates an SSVT. See in “REQUEST=CREATE parameter of IEFSSVT” on page 861 for the syntax of this request.
- IEFSSVT REQUEST=DISABLE, which disables supported function codes. See in “REQUEST=DISABLE parameter of IEFSSVT” on page 865 for the syntax of this request.
- IEFSSVT REQUEST=ENABLE, which enables additional function codes. See in “REQUEST=ENABLE parameter of IEFSSVT” on page 868 for the syntax of this request.
- IEFSSVT REQUEST=EXCHANGE, which replaces the function routine associated with a supported function code with another function routine. See in “REQUEST=EXCHANGE parameter of IEFSSVT” on page 872 for the syntax of this request.

For ease of use, the standard form of the macro is shown for each IEFSSVT request. The requests are described on the following pages along with the:

- Standard form syntax diagram
- Description of the parameters

The following information is described once at the beginning of the IEFSSVT macro description:

- Environment
- Programming requirements
- Restrictions
- Input register information
- Output register information
- Performance implications

Following the descriptions of the standard forms of all requests are:

- Abend codes
IEFSSVT macro

- Return and reason codes
- Examples

The REQUEST=CREATE, REQUEST=DISABLE, REQUEST=ENABLE, and REQUEST=EXCHANGE parameters are mutually exclusive. You can select only one.

For information about using dynamic subsystem services, see z/OS MVS Using the Subsystem Interface. This topic also includes information about related macros IEFSSVTI and IEFSSI.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>- Supervisor state</td>
</tr>
<tr>
<td></td>
<td>- Any system key</td>
</tr>
<tr>
<td></td>
<td>- PSW key mask (PKM) allowing key 0-7</td>
</tr>
<tr>
<td></td>
<td>- APF authorization</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24-bit or 31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or Access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space</td>
</tr>
</tbody>
</table>

Programming requirements

Before invoking IEFSSVT, you must invoke IEFSSVTI to build a table of function routines and function codes as input to IEFSSVT.

Also:
- Include the CVT and IEFJESCT mapping macros in your program.
- Include the IEFJSRC mapping macro in your program. This macro defines the dynamic SSI return and reason codes.

Restrictions

The services that IEFSSVT provides are available only to dynamic subsystems, which are those subsystems that have been defined to the SSI in one of the following ways:
- Processing the keyword format of the IEFSSNxx parmlib member during IPL
- Issuing the IEFSSI macro
- Issuing the SETSSI system command.

Input register information

Before issuing the IEFSSVT macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.
Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

REQUEST=CREATE parameter of IEFSSVT

The IEFSSVT macro with the CREATE parameter builds the SSVT. An SSVT built with the IEFSSVT REQUEST=CREATE is referred to as an SSI-managed vector table. Only SSI-managed SSVTs can take advantage of the dynamic SSI services. See z/OS MVS Using the Subsystem Interface for more information about dynamic SSI services.

Syntax for REQUEST=CREATE

The syntax of the IEFSSVT REQUEST=CREATE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>; symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSVT.</td>
</tr>
<tr>
<td>IEFSSVT</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFSSVT.</td>
</tr>
</tbody>
</table>
### IEFSSVT macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBNAME=subname</td>
<td></td>
</tr>
<tr>
<td>REQUEST=CREATE</td>
<td></td>
</tr>
<tr>
<td>SSVTDATA=ssvtdata</td>
<td>ssvtdata: RS-type address</td>
</tr>
<tr>
<td>OUTTOKEN=outtoken</td>
<td>outtoken: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>SUBPOOL=subpool</td>
<td>subpool: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>SUBPOOL=241</td>
<td>Default: SUBPOOL=241</td>
</tr>
<tr>
<td>MAXENTRIES=maxentries</td>
<td>maxentries: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>LOADTOGLOBAL=NO</td>
<td>Default: LOADTOGLOBAL=NO</td>
</tr>
<tr>
<td>LOADTOGLOBAL=YES</td>
<td></td>
</tr>
<tr>
<td>ERRFUNCT=errfunct</td>
<td>errfunct: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>PLISTVER=IMPLIED_VERSION</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>PLISTVER=plistver</td>
<td>plistver: 1</td>
</tr>
<tr>
<td>RETCODE=rcode</td>
<td>rcode: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>RSNCODE=rscode</td>
<td>rscode: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>COM=com</td>
<td>com: comment string</td>
</tr>
<tr>
<td>COM=NULL</td>
<td>Default: COM=NULL</td>
</tr>
<tr>
<td>MF=S</td>
<td></td>
</tr>
<tr>
<td>MF=(L,list addr)</td>
<td>list addr: symbol.</td>
</tr>
<tr>
<td>MF=(L,list addr,attr)</td>
<td>attr: 1- to 60-character input string.</td>
</tr>
<tr>
<td>MF=(L,list addr,0D)</td>
<td>Default: 0D</td>
</tr>
<tr>
<td>MF=(E,list addr)</td>
<td>list addr: RS-type address or address in register (1) - (12) of a storage area.</td>
</tr>
<tr>
<td>MF=(E,list addr,COMPLETE)</td>
<td>Default: COMPLETE</td>
</tr>
<tr>
<td>MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>
Parameters for REQUEST=CREATE

The parameters are explained as follows:

**SUBNAME**=subname

A required 4-character parameter that specifies the field (or an address in a register) containing the subsystem name. It must be the name of a subsystem that has been previously defined to the system using dynamic SSI services. See z/OS MVS Using the Subsystem Interface for more information about dynamic SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

**REQUEST**=CREATE

A parameter that specifies an SSVT is to be built and initialized.

**SSVTDATA**=ssvtdata

A required parameter that specifies the name of the function routine input table that associates supported SSVT function codes with function routines. Use the IEFSSVTI macro to build the table. Use this field to contain the name specified by the IEFSSVTI SSVTDATA parameter.

**OUTTOKEN**=outtoken

A required 32-bit parameter that specifies the name (or an address in a register) of an output token used to identify the SSVT.

**SUBPOOL**=subpool

An optional byte parameter that specifies the name (or an address in a register) of an input field that contains the subpool number in which the SSVT is to be built. The subpool must represent common storage. The default is 241.

**MAXENTRIES**=maxentries

A required halfword parameter that specifies the name (or an address in a register) of an input field that defines the maximum number of function routine entries that the SSVT can contain. The maximum number must be:

- Greater than or equal to 1 and less than or equal to 255
- Greater than or equal to the number of function routines defined in the input table created with the IEFSSVTI macro.
- The maximum number of function routine entries the calling subsystem requires for the SSVT.

**Note:** Consider that the value for MAXENTRIES should provide for additional function routines that can be defined through the SET or EXCHANGE parameters of the IEFSSVT service.

**LOADTOGLOBAL**=NO

**LOADTOGLOBAL**=YES

An optional parameter that specifies that the function routines are to be loaded to global storage. Use this parameter when the function routines:

- Are specified by name in the function routine input table that the IEFSSVTI macro created
- Do not reside in the link pack area.

This parameter is ignored if the input function routines are specified by address.

The meanings are:
IEFSSVT macro

**NO** Indicates that a load-to-global is not necessary for the function routines. No is the default.

**YES**
Indicates that a load-to-global is necessary for the function routines. The SSI issues a LOAD for each named function routine with the following parameters:
- EOM=YES
- LSEARCH=NO
- GLOBAL=(YES,P)

,**ERRFUNCT=errfunct**
An optional 8-character parameter that specifies the name (or an address in a register) of an output field that receives the function routine name being processed when an error occurred. Check this output field if you get return code IEFSSVT_LOAD_ERROR (decimal 16) from the IEFSSVT macro.

,**PLISTVER=IMPLIED_VERSION**
,**PLISTVER=MAX**
,**PLISTVER=plistver**
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

**IMPLIED_VERSION**
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

**MAX**
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1 The currently available parameters.

To code: specify in this input parameter one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 1

,**RETCODE=retcode**
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the return code. The return code is copied from general purpose register 15.

,**RSNCODE=rsncode**
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the reason code. The reason code is copied from general purpose register 0.
IEFSSVT macro

,COM=com
,COM=NULL

An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,list addr)
,MF=(L,list addr,attr)
,MF=(L,list addr,0D)
,MF=(E,list addr)
,MF=(E,list addr,COMPLETE)

Use MF=S to specify the standard form of the IEFSSVT macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSVT macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use the MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSVT macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1-to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=DISABLE parameter of IEFSSVT

The IEFSSVT macro with the DISABLE parameter disables supported function codes.

Syntax for REQUEST=DISABLE

The syntax of the IEFSSVT REQUEST=DISABLE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSVT.</td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEFSSVT</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFSSVT.</td>
</tr>
<tr>
<td>SUBNAME=asename</td>
<td></td>
</tr>
<tr>
<td>,REQUEST=DISABLE</td>
<td></td>
</tr>
<tr>
<td>,SSVTDATA=ssvtdata</td>
<td>ssvtdata: RS-type address</td>
</tr>
<tr>
<td>,INTOKEN=intoken</td>
<td>intoken: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,INTOKEN=NULL</td>
<td>Default: INTOKEN=NULL</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=plistver</td>
<td>plistver: 1</td>
</tr>
<tr>
<td>,RETCODE=rcode</td>
<td>recode: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,COM=com</td>
<td>com: comment string</td>
</tr>
<tr>
<td>,COM=NULL</td>
<td>Default: COM=NULL</td>
</tr>
<tr>
<td>,MF=S</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td>list addr: symbol.</td>
</tr>
<tr>
<td>,MF=(L,list addr,attr)</td>
<td>attr: 1- to 60-character input string.</td>
</tr>
<tr>
<td>,MF=(L,list addr,0D)</td>
<td>Default: 0D</td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td>list addr: RS-type address or address in register (1) - (12) of a storage area.</td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td>Default: COMPLETE</td>
</tr>
<tr>
<td>,MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters for REQUEST=DISABLE

The parameters are explained as follows:

- **SUBNAME=asename**
  A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.
This field must be padded to the right with blanks or nulls if it less than 4 characters long.

\textbf{REQUEST=DISABLE}

A parameter that specifies that you want to disable function codes.

\textbf{SSVTDATA=ssvtdata}

A required parameter that specifies the name of the function routine input table that identifies the function codes that you want to disable. The IEFSSVTI macro has built the table. This field contains the name specified by the IEFSSVTI SSVTDATA parameter.

For the disable request, the system uses only the function code information. The function routine names or addresses in the input table are ignored.

\textbf{INTOKEN=intoken}

\textbf{INTOKEN=NULL}

An optional 32-bit parameter that specifies the name (or an address in a register) of an input token that represents the target SSVT (OUTTOKEN from REQUEST=CREATE). The default is NULL.

\textbf{PLISTVER=IMPLIED_VERSION}

\textbf{PLISTVER=MAX}

\textbf{PLISTVER=plistver}

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

\textbf{IMPLIED_VERSION}

The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

\textbf{MAX}

The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

\textbf{1}

The currently available parameters.

\textbf{To code:} specify in this input parameter one of the following:

\begin{itemize}
  \item IMPLIED_VERSION
  \item MAX
  \item A decimal value of 1
\end{itemize}

\textbf{RETCODE=retcode}

An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the return code. The return code is copied from general purpose register 15.
RSNCODE=rsncode
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the reason code. The reason code is copied from general purpose register 0.

COM=com
COM=NULL
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

MF=S
MF=(L,list addr)
MF=(L,list addr,attr)
MF=(L,list addr,0D)
MF=(E,list addr)
MF=(E,list addr,COMPLETE)
Use MF=S to specify the standard form of the IEFSSVT macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSVT macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use the MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSVT macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

list addr
A required parameter that specifies the name of a storage area for the parameter list.

attr
An optional 1-to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=ENABLE parameter of IEFSSVT
The IEFSSVT macro with the ENABLE parameter activates new function codes or reactivates previously disabled function codes. You can enable new function codes only if the SSVT has available function routine slots to contain any new function routines. An ENABLE request may not need to specify new function routines, if the routine that supports a new code is already supporting a previously enabled code.
Syntax for **REQUEST=ENABLE**

The syntax of the IEFSSVT REQUEST=ENABLE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>IEFSSVT</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSVT.</td>
</tr>
<tr>
<td>SUBNAME=subname</td>
<td></td>
</tr>
<tr>
<td>,REQUEST=ENABLE</td>
<td></td>
</tr>
<tr>
<td>,SSVTDATA=svtdata</td>
<td><em>svtdata</em>: RS-type address</td>
</tr>
<tr>
<td>,INTOKEN=intoken</td>
<td><em>intoken</em>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,INTOKEN=NULL</td>
<td><strong>Default</strong>: INTOKEN=NULL</td>
</tr>
<tr>
<td>,LOADTOGLOBAL=NO</td>
<td><strong>Default</strong>: LOADTOGLOBAL=NO</td>
</tr>
<tr>
<td>,LOADTOGLOBAL=YES</td>
<td></td>
</tr>
<tr>
<td>,ERRFUNCT=errfunct</td>
<td><em>errfunct</em>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED VERSION</td>
<td><strong>Default</strong>: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=listver</td>
<td><em>listver</em>: 1</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td><em>retcode</em>: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,RSNCODE=rncode</td>
<td><em>rncode</em>: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,COM=com</td>
<td><em>com</em>: comment string</td>
</tr>
<tr>
<td>,COM=NULL</td>
<td><strong>Default</strong>: COM=NULL</td>
</tr>
<tr>
<td>,MF=S</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td><em>list addr</em>: symbol.</td>
</tr>
<tr>
<td>,MF=(L,list addr,attr)</td>
<td><em>attr</em>: 1- to 60-character input string.</td>
</tr>
</tbody>
</table>
### IEFSSVT macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,MF=(L,list addr,0D)</td>
<td>Default: 0D</td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td>list addr: RS-type address or address in register (1) - (12) of a storage area.</td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td>Default: COMPLETE</td>
</tr>
<tr>
<td>,MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters for REQUEST=ENABLE

The parameters are explained as follows:

**SUBNAME=subname**

A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services. This field must be padded to the right with blanks or nulls if it less than 4 characters long.

**REQUEST=ENABLE**

A parameter that specifies that you want to enable the function codes specified in the SSVTDATA parameter.

**SSVTDATA=ssvtdata**

A required parameter that specifies the name of the function routine input table that identifies the new function codes that the SSVT supports and their related function routines. Use the IEFSSVTI macro to build the table. This field contains the the name specified by the IEFSSVTI SSVTDATA parameter.

**INTOKEN=intoken**

An optional 32-bit parameter that specifies the name (or an address in a register) of an input token that represents the target SSVT. The default is NULL.

**LOADTOGLOBAL=NO**

**LOADTOGLOBAL=YES**

An optional parameter that specifies that the function routines are to be loaded to global storage. Use this parameter when the function routines:

- Are specified by name in the function routine input table that the IEFSSVTI macro created
- Do not reside in the link pack area.

This parameter is ignored if the input function routines are specified by address.

The meanings are:

- **NO** Indicates that a load-to-global is not necessary for the function routines. This is the default.
- **YES** Indicates that a load-to-global is necessary for the function routines. The SSI issues a LOAD for the named function routine with the following parameters:
  - EOM=YES
  - LSEARCH=NO
GLOBAL=(YES,P)

,ERRFUNCT=errfunct
An optional 8-character parameter that specifies the name (or an address in a
register) of an output field that receives the function routine name being
processed when an error occurred. Check this output field if you get return
code IEFSSVT_LOAD_ERROR (decimal 22) from the IEFSSVT macro.

PLISTVER=IMPLIED_VERSION
PLISTVER=MAX
PLISTVER=plistver
Specifies the version of the macro. PLISTVER determines which parameter list
the system generates. PLISTVER is an optional input parameter on all forms of
the macro, including the list form. When using PLISTVER, specify it on all
macro forms used for a request and with the same value on all of the macro
forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be
processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the
default.

MAX
The largest size parameter list currently possible. This size might grow
from release to release and affect the amount of storage that your program
needs.

If you can tolerate the size change, IBM recommends that you always
specify PLISTVER=MAX on the list form of the macro. Specifying MAX
ensures that the list-form parameter list is always long enough to hold all
the parameters you might specify on the execute form; in this way, MAX
ensures that the parameter list does not overwrite nearby storage.

1 The currently available parameters.

To code: specify in this input parameter one of the following:
• IMPLIED_VERSION
• MAX
• A decimal value of 1

,RETCODE=retcode
An optional 4-byte parameter that specifies a name (or register) of an output
field where the system places the return code. The return code is copied from
general purpose register 15.

,RSNCODE=rsncode
An optional 4-byte parameter that specifies a name (or register) of an output
field where the system places the reason code. The reason code is copied from
general purpose register 0.

,COM=com
,COM=DEFAULT
An optional parameter that specifies the character input that appears in the
block comment before the macro invocation. Use it to make comments about
the macro invocation. The comment must be enclosed in quotation marks if it
contains any lower case characters. The default is DEFAULT.

,MF=S
,MF=(L,list addr)
,MF=(L,list addr,attr)
IEFSSVT macro

MF=(L,list addr,0D)
MF=(E,list addr)
MF=(E,list addr,COMPLETE)

Use MF=S to specify the standard form of the IEFSSVT macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSVT macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use the MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSVT macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,LIST addr
A required parameter that specifies the name of a storage area for the parameter list.

,ATTR
An optional 1-to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code ATTR, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=EXCHANGE parameter of IEFSSVT

The IEFSSVT macro with the EXCHANGE parameter exchanges the function routine that supports a function code with a different function routine.

Syntax for REQUEST=EXCHANGE

The syntax of the IEFSSVT REQUEST=EXCHANGE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSVT.</td>
</tr>
<tr>
<td>IEFSSVT</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFSSVT.</td>
</tr>
<tr>
<td>SUBNAME=subname</td>
<td></td>
</tr>
</tbody>
</table>
## Parameters for REQUEST=EXCHANGE

The parameters are explained as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,REQUEST=EXCHANGE</td>
<td></td>
</tr>
<tr>
<td>,SSVTDATA=ssvtdata</td>
<td>ssvtdata: RS-type address</td>
</tr>
<tr>
<td>,INTOKEN=intoken</td>
<td>intoken: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,INTOKEN=NULL</td>
<td>Default: INTOKEN=NULL</td>
</tr>
<tr>
<td>,LOADTOGLOBAL=NO</td>
<td>Default: LOADTOGLOBAL=NO</td>
</tr>
<tr>
<td>,LOADTOGLOBAL=YES</td>
<td></td>
</tr>
<tr>
<td>,ERRFUNCT=errfunct</td>
<td>errfunct: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=plistver</td>
<td>plistver: 1</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,COM=com</td>
<td>com: comment string</td>
</tr>
<tr>
<td>,COM=NULL</td>
<td>Default: COM=NULL</td>
</tr>
<tr>
<td>,MF=S</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td>list addr: symbol.</td>
</tr>
<tr>
<td>,MF=(L,list addr,attr)</td>
<td>attr: 1- to 60-character input string.</td>
</tr>
<tr>
<td>,MF=(L,list addr,0D)</td>
<td>Default: 0D</td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td>list addr: RS-type address or address in register (1) - (12) of a storage area.</td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td>Default: COMPLETE</td>
</tr>
<tr>
<td>,MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>

### SUBNAME=subname

A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it less than 4 characters long.
IEFSSVT macro

,,REQUEST=EXCHANGE
   A parameter that specifies that you want to exchange existing function routines
   with new function routines.

,,SSVTDATA=ssvtdata
   A required parameter that specifies the name of the function routine input
   table that identifies the new function codes affected by the exchange and the
   new function routines that support them. Use the IEFSSVTI macro to build the
   table. This field contains the the name specified by the IEFSSVTI SSVTDATA
   parameter.

,,INTOKEN=intoken
,,INTOKEN=NULL
   An optional 32-bit parameter that specifies the name (or an address in a
   register) of an input token that represents the target SSVT. The default is
   NULL.

,,LOADTOGLOBAL=NO
,,LOADTOGLOBAL=YES
   An optional parameter that specifies that the function routines are to be loaded
   to global storage. Use this parameter when the function routines:
   - Are specified by name in the function routine input table that the IEFSSVTI
     macro created
   - Do not reside in the link pack area.
   This parameter is ignored if the input function routines are specified by
   address.
   NO Indicates that a load-to-global is not necessary for the function routines.
   This is the default.
   YES Indicates that a load-to-global is necessary for the function routines. The
   SSI issues a LOAD for the named function routine with the following
   parameters:
   - EOM=YES
   - LSEARCH=NO
   - GLOBAL=(YES,P)

,,ERRFUNCT=errfunct
   An optional 8-character parameter that specifies the name (or an address in a
   register) of an output field that receives the function routine name being
   processed when an error occurred. Check this output field if you get return
   code IEFSSVT_LOAD_ERROR (decimal 22) from the IEFSSVT macro.

PLISTVER=IMPLIED_VERSION
PLISTVER=MAX
PLISTVER=plistver
   Specifies the version of the macro. PLISTVER determines which parameter list
   the system generates. PLISTVER is an optional input parameter on all forms of
   the macro, including the list form. When using PLISTVER, specify it on all
   macro forms used for a request and with the same value on all of the macro
   forms. The values are:
   IMPLIED_VERSION
   The lowest version that allows all parameters specified on the request to be
   processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the
   default.
**IEFSSVT macro**

**MAX**
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1 The currently available parameters.

To code: specify in this input parameter one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 1

**RETCODE=retcode**
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the return code. The return code is copied from general purpose register 15.

**RSNCODE=rsncode**
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the reason code. The reason code is copied from general purpose register 0.

**COM=com**
**COM=NULL**
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

**MF=S**
**MF=(L,list addr)**
**MF=(L,list addr,attr)**
**MF=(L,list addr,0D)**
**MF=(E,list addr)**
**MF=(E,list addr,COMPLETE)**
Use MF=S to specify the standard form of the IEFSSVT macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSVT macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use the MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSVT macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

**list addr**
A required parameter that specifies the name of a storage area for the parameter list.
**IEFSSVT macro**

`attr`
An optional 1-to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code `attr`, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

`COMPLETE`
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

**ABEND codes**
An invocation of the IEFSSVT macro may result in an abend code X'8C5'. See z/OS MVS System Codes for an explanation of this abend code.

**Return and reason codes**
When the IEFSSVT macro returns control to your program, GPR 15 (and `retcode`, if you coded RETCODE) contains a return code. When the value in GPR 15 is not 0, GPR 0 (and `rsncode` if you coded RSNCODE) contains the reason code.

The IEFJSRC mapping macro provides equate symbols for the return and reason codes. The equate symbols associated with each return code are:

<table>
<thead>
<tr>
<th>Return Code Decimal</th>
<th>Equate Symbol for Return Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>IEFSSVT_SUCCESS</td>
</tr>
<tr>
<td>04 (04)</td>
<td>IEFSSVT_WARNING</td>
</tr>
<tr>
<td>08 (08)</td>
<td>IEFSSVT_INVALID_PARAMETERS</td>
</tr>
<tr>
<td>12 (0C)</td>
<td>IEFSSVT_REQUEST_FAIL</td>
</tr>
<tr>
<td>16 (010)</td>
<td>IEFSSVT_LOAD_ERROR</td>
</tr>
<tr>
<td>20 (014)</td>
<td>IEFSSVT_SYSTEM_ERROR</td>
</tr>
<tr>
<td>24 (018)</td>
<td>IEFSSVT_UNAVAILABLE</td>
</tr>
</tbody>
</table>

The following table contains return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

**Table 76. Return and Reason Codes for the IEFSSVT Macro**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>00 (00)</td>
<td>Equate Symbol: IEFSSVT_FUNCTIONS_COMPLETE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The request completed successfully. The result depends on the request:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CREATE — An SSVT is created.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DISABLE — Supported function codes are disabled.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ENABLE — Additional function codes are enabled.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EXCHANGE — New function routines replace supported function routines.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None.</td>
<td></td>
</tr>
<tr>
<td>04 (04)</td>
<td>00 (00)</td>
<td>Equate Symbol: None</td>
<td></td>
</tr>
</tbody>
</table>
### Table 76. Return and Reason Codes for the IEFSSVT Macro (continued)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>08 (08)</td>
<td>00 (00)</td>
<td>Equate Symbol: IEFSSVT_SUBSYSTEM_UNKNOWN Meaning: The subsystem was not defined to the SSI Action: None.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>04 (004)</td>
<td>Equate Symbol: IEFSSVT_NON_DYNAMIC Meaning: The subsystem is not a dynamic subsystem Action: None.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>08 (008)</td>
<td>Equate Symbol: IEFSSVT_BAD_VT_TOKEN Meaning: The SSVT token does not correspond to a valid SSVT Action: None.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>12 (00C)</td>
<td>Equate Symbol: IEFSSVT_INVALID_NAME Meaning: The subsystem name or the routine name contains characters that are not valid Action: None.</td>
</tr>
<tr>
<td>08 (08)</td>
<td>16 (010)</td>
<td>Equate Symbol: IEFSSVT_INVALID_FUNCTION_CODE Meaning: The function code is outside the valid range. Action: Correct the function code.</td>
</tr>
<tr>
<td>08 08</td>
<td>20 (014)</td>
<td>Equate Symbol: IEFSSVT_DUPLICATE_FUNCTION_CODE Meaning: The function code appears more than once in the function routine input table. Action: Delete the duplicate specification.</td>
</tr>
<tr>
<td>08 08</td>
<td>24 (018)</td>
<td>Equate Symbol: IEFSSVT_INVALID_FROUTINE Meaning: The function routine name or address is null. Action: If working with a function routine input table in dynamic storage, use the IEFSSVTI SET function to identify the function routine by name or address. If using a static table, the function routine must be identified by name using the IEFSSVTI ENTRY function.</td>
</tr>
<tr>
<td>08 08</td>
<td>28 (01C)</td>
<td>Equate Symbol: IEFSSVT_NO_FCODES Meaning: The function routine entry in the function routine input table does not specify any function codes. Action: Specify the function codes with either the IEFSSVTI ENTRY or SET function.</td>
</tr>
<tr>
<td>12 (0C)</td>
<td>00 (00)</td>
<td>Equate Symbol: None</td>
</tr>
<tr>
<td>12 (0C)</td>
<td>100 (064)</td>
<td>Equate Symbol: IEFSSVT_MAX VECTOR_TABLES Meaning: The maximum number of SSVTs already exists for the subsystem. This is a CREATE request error. Action: Additional vector tables cannot be created during the current IPL. Use the IEFSSVT ENABLE and DISABLE services to modify the response of an existing vector table.</td>
</tr>
</tbody>
</table>
Table 76. Return and Reason Codes for the IEFSSVT Macro (continued)

|--------------------------|--------------------------|-------------------------------|--------------------|
| 12 (0C)                  | 101 (065)                | IEFSSVT_STORAGE              | Meaning: Unable to obtain storage for an SSVT. This is a CREATE request error.  
Action: Review the use of system storage to determine why sufficient storage was not available. Try the request again later. |
|                         | 102 (066)                | IEFSSVT_MAXENTRIES_TOO_SMALL | Meaning: The MAXENTRIES value is less than the number of function routines in the function routine input table. This is a CREATE request error.  
Action: Correct the MAXENTRIES value. It must be at least as large as the number of function routines represented in the function routine input table. |
|                         | 103 (067)                | IEFSSVT_MAXENTRIES_TOO_BIG   | Meaning: The MAXENTRIES value is greater than the maximum value (255). This is a CREATE request error.  
Action: None. |
| 12 (0C)                  | 200 (0C8)                | IEFSSVT_ENABLE_NO_ELIGIBLE_VT | Meaning: The SSVT is not specified and a valid default is not available. This is an ENABLE request error.  
Action: Use the IEFSSVT CREATE function to create an SSI-managed vector table. |
|                         | 201 (0C9)                | IEFSSVT_ENABLE_MAX_ROUTINES  | Meaning: The SSVT does not have available space for new function routines. This is an ENABLE request error.  
Action: If the subsystem has only one SSI-managed vector table, use the IEFSSVT CREATE function to create a second larger one, which responds to all required function codes.  
If the subsystem already has two vector tables, the problem cannot be corrected without re-IPLing the system, unless some function codes can be disabled to make room for the new required function routines. |
|                         | 202 (0CA)                | IEFSSVT_FUNCTION_ALREADY_ENABLED | Meaning: The subsystem already responds to one of the function codes for which this request was submitted. This an ENABLE request error.  
Action: If you want to change the routine that supports the function, use the IEFSSVT EXCHANGE function. |
| 12 (0C)                  | 300 (12C)                | IEFSSVT_DISABLE_NO_ELIGIBLE_VT | Meaning: The SSVT is not specified and a valid default is not available. This an DISABLE request error.  
Action: Create an SSI-managed SSVT using the IEFSSVT CREATE function. |
### Table 76. Return and Reason Codes for the IEFSSVT Macro (continued)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12 (0C)</td>
<td>500 (1F4)</td>
<td>IEFSSVT_EXCHANGE_NO_ELIGIBLE_VT</td>
<td>The SSVT is not specified and a valid default is not available. This is an EXCHANGE request error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Action</strong>: Create an SSI-managed SSVT using the IEFSSVT CREATE function.</td>
</tr>
<tr>
<td>12 (0C)</td>
<td>501 (1F5)</td>
<td>IEFSSVT_EXCHANGE_MAX_ROUTINES</td>
<td>The SSVT does not have available space for new function routines. This is an EXCHANGE request error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Action</strong>: If the subsystem has only one SSI-managed vector table, use the IEFSSVT CREATE function to create a second larger one, which responds to all required function codes. If the subsystem already has two vector tables, the problem cannot be corrected without re-IPLing the system, unless some function codes can be disabled to make room for the new required function routines.</td>
</tr>
</tbody>
</table>

### Examples

For the following examples, assume that the function routine input tables have already been built using the IEFSSVTI macro.

#### Example 1
Create an SSVT, reserving space for 5 function routines. The function routines reside in LPA, so the LOADTOGLOBAL parameter is allowed to default to NO.

```
IEFSSVT REQUEST=CREATE, SUBNAME=SNAME, SSVTDATA=FROUTINE_TABLE, X
  MAXENTRIES=5, OUTTOKEN=NEWTOKEN, ERRFUNCT=BADNAME, X
  RETCODE=RETURN_CODE, RSNCODE=REASON_CODE...
```

```
SNAME DC CL4 'FRED'
WORKAREA DSECT OF
NEWTOKEN DS CL4
BADNAME DS CL8
RETURN_CODE DS F
REASON_CODE DS F
WORKLEN EQU *-WORKAREA
```

#### Example 2
Enable an additional function code in the SSVT that was created in example 1. The function routine input table ENABLE_TABLE describes only the new function code and its associated function routine.

```
IEFSSVT REQUEST=ENABLE, SUBNAME=SNAME, SSVTDATA=ENABLE_TABLE, X
  INTOKEN=NEWTOKEN, ERRFUNCT=BADNAME, X
  RETCODE=RETURN_CODE, RSNCODE=REASON_CODE
```

#### Example 3
Disable one of the function codes currently supported by the SSVT created in example 1. The function routine input table DISABLE_TABLE describes only the function code to be disabled. It does not have to provide function routine information.
Example 4

Change the function routine that responds to one of the function codes supported by the SSVT in example 1. The function routine input table EXCHANGE_TABLE identifies the function code and the new function routine.

IEFSSVT REQUEST=EXCHANGE, SUBNAME=SNAME, SSVTDATA=EXCHANGE_TABLE, INTOKEN=NEWTOKEN, ERRFUNCT=BADNAME, RETCODE=RETURN_CODE, RSNCODE=REASON_CODE
Chapter 82. IEFSSVTI — Associate function routines with function codes

Description

Use the IEFSSVTI macro to create a table that relates function routines to the function codes they support. This information is passed to the IEFSSVT macro as input when creating or modifying an SSVT.

The IEFSSVTI macro:
- Creates a static function routine input table
- Reserves dynamic storage for a function routine input table
- Copies a static table to dynamic storage
- Modifies a function routine input table in dynamic storage

A static function routine input table is a table that does not change at program run time and is used when all the information, such as the function routine names, are known at compile time.

The IEFSSVTI macro does not attempt to verify that its caller is a dynamic subsystem. IEFSSVTI can be used only with IEFSSVT.

Each function routine defined by the SET or ENTRY parameters of the IEFSSVTI macro occupies a separate entry in the SSVT. The SSVT size is limited to the number of entries specified by the MAXENTRIES parameter at the time when the SSVT is created (IEFSSVT CREATE). As a result, IBM suggests economizing the use of the slots by identifying only unique function routines for each SET or ENTRY request of IEFSSVTI. In this way, if a function routine is common to several function codes, a single call is made to IEFSSVTI to relate all of the function codes to the function routine, rather than calling IEFSSVTI many times, relating the same function routine to many function codes. For examples of relating multiple function codes to a single function routine, see "Example 1" on page 884, which identifies the LISTEN function routine related to two function codes, or see "Example 2" on page 884, which identifies the VERSION function routine, related to two function codes.

Note: The IEFSSVTI macro expands in-line and therefore does not impose restrictions on authorization, dispatch mode, cross-memory mode, locks or control parameters. The program using the table created by the IEFSSVTI macro is subject to the restrictions of the IEFSSVT macro.

The types for the macro are:
- IEFSSVTI TYPE=LIST, which creates a DSECT that maps the format of the function routine input table. See in "TYPE=LIST parameter of IEFSSVTI" on page 885 for the syntax of this request.
- IEFSSVTI TYPE=INITIAL, which begins the definition of a static function routine input table. See in "TYPE=INITIAL parameter of IEFSSVTI" on page 886 for the syntax of this request.
- IEFSSVTI TYPE=ENTRY, which defines a function routine entry in a static input table. See in "TYPE=ENTRY parameter of IEFSSVTI" on page 887 for the syntax of this request.
IEFSSVTI macro

- IEFSSVTI TYPE=FINAL, which ends the definition of a static function routine input table. See in “TYPE=FINAL parameter of IEFSSVTI” on page 889 for the syntax of this request.
- IEFSSVTI TYPE=SET, which modifies a function routine entry in an existing input table. See in “TYPE=SET parameter of IEFSSVTI” on page 889 for the syntax of this request.
- IEFSSVTI TYPE=RESERVE, which reserves storage for a function routine input table. See in “TYPE=RESERVE parameter of IEFSSVTI” on page 892 for the syntax of this request.
- IEFSSVTI TYPE=COPY, which copies a function routine input table. See in “TYPE=COPY parameter of IEFSSVTI” on page 893 for the syntax of this request.

For ease of use, the standard form of the macro is shown for each IEFSSVTI type. The types are described on the following pages along with the:
- Standard form syntax diagram
- Description of the parameters

The following information is described once at the beginning of the IEFSSVTI macro description:
- Environment
- Programming requirements
- Restrictions
- Input register information
- Output register information
- Performance implications

Following the descriptions of the standard forms of all requests are:
- Abend codes
- Return and reason codes
- Examples

The TYPE=LIST, TYPE=INITIAL, TYPE=ENTRY, TYPE=FINAL, TYPE=SET, TYPE=RESERVE, and TYPE=COPY parameters are mutually exclusive. You can select only one.

For information about using dynamic subsystem services, see z/OS MVS Using the Subsystem Interface. This topic also includes information about the related macro IEFSSI.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state with any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24-bit or 31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or Access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space</td>
</tr>
</tbody>
</table>
Programming requirements

If the subsystem function routines are identified by address using the FUNCADDR parameter, the invoking program must load the function routines by using the LOAD macro, or obtain their addresses before invoking this macro.

Restrictions

The input table this macro creates:

- Can only be used with the IEFSSVT macro
- Cannot be used by the IEFJSVEC service to create SSVTs

The register form can be used to specify macro parameter variables only in TYPE=SET invocations.

Input register information

Before issuing the IEFSSVTI macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

ABEND codes

None. The IEFSSVT macro indicates whether the IEFSSVTI macro processing was successful, because you must use the IEFSSVT macro with the IEFSSVT macro when creating an SSVT.

Return and reason codes

None.
Examples

The following notes apply to the supplied examples:

Note:
1. A set of IEFSSVTI macro invocations, beginning with TYPE=INITIAL and ending with TYPE=FINAL must contain some invocations specifying FUNCNAME and some specifying FUNCADDR. The input table does not have to identify all function routines the same way.

2. Usage scenarios

   When you know all of the information at compile time, you can create a static function routine input table using: TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL.

   When you do not know all of the information at compile time, you can create a dynamic function routine input table as follows:
   - For reentrant programs: create a static function routine input table, reserve storage for a dynamic table of the same size using TYPE=RESERVE, copy the static table to the dynamic table using TYPE=COPY, and modify the dynamic table using TYPE=SET.
   - For non-reentrant programs: create a static table and modify it using TYPE=SET, or copy a static table to dynamic storage and modify the dynamic table using TYPE=SET.

Example 1

Build a static function routine input table, specifying the function routines by name.

```
IEFSSVTI TYPE=INITIAL, SSVTDATA=MY_STAT_TABLE, +
   TABLEN=STAT_TABLE_LENGTH
IEFSSVTI TYPE=ENTRY, FUNCNAME=LISTEN, NUMFCODES=2, +
   FCODES=(SSOBWTO, SSOBWTL)
IEFSSVTI TYPE=ENTRY, FUNCNAME=VERSION, NUMFCODES=1, +
   FCODES=SSOBSSVI
IEFSSVTI TYPE=FINAL
```

Example 2

Build a dynamic function routine input table, specifying function routines by address. A static function routine input table is defined as a template and copied to dynamic storage reserved by an IEFSSVTI TYPE=RESERVE invocation. The dynamic function routine input table is completed using TYPE=SET invocation.

INITRTN CSECT

```
*:***********************************************************************
*: LOAD the function routines, store the entry point addresses,
*: and DELETE.
*:***********************************************************************

LOAD EP=LISTEN
ST R0, LISTEN_ADDR
DELETE EP=LISTENER

*:***********************************************************************
*: LOAD EP=VERSION
ST R0, VERSION_ADDR
DELETE EP=VERSION

*:***********************************************************************
*: Copy the static table to dynamic storage
*:***********************************************************************

IEFSSVTI TYPE=COPY, SSVTDATA=MY_DYN_TABLE, +
   SOURCE=MY_TABLE

*
IEFSSVTI macro

******************************************************************************
* Set the function routine address information in the input
* table. Override the function codes supported for the first
* entry, so that only SSOBWTO is supported and not SSOBWL.
******************************************************************************
IEFSSVTI TYPE=SET,SSVTDATA=MY_DYN_TABLE,SOURCE=MY_TABLE, +
   ENTRYDATA=1,FUNCADDR=LISTEN_ADDR,FCODES=SSOBWTO
*     IEFSSVTI TYPE=SET,SSVTDATA=MY_DYN_TABLE,SOURCE=MY_TABLE, +
     ENTRYDATA=2,FUNCADDR=VERSION_ADDR...
******************************************************************************

* Working storage
******************************************************************************
WORKAREA DSECT 0F
LISTEN_ADDR DS A Address of listen function +
VERSION_ADDR DS A Address of version info function +
******************************************************************************
* Reserve storage for dynamic function routine input table
******************************************************************************
IEFSSVTI TYPE=RESERVE,SSVTDATA=MY_DYN_TABLE, +
   TABLEN=STAT_TABLE_LENGTH
*     WORKLEN EQU */-WORKAREA Length of work area
*     IEFSSVTI TYPE=LIST Generate table mappings
*     INITRTN CSECT
*     *
* Create static function routine input table
******************************************************************************
IEFSSVTI TYPE=INITIAL,SSVTDATA=MY_STAT_TABLE, +
   TABLEN=STAT_TABLE_LENGTH
IEFSSVTI TYPE=ENTRY,NUMFCODES=2,FCODES=(SSOBWTO,SSOBWTL)
IEFSSVTI TYPE=ENTRY,NUMFCODES=1,FCODES=SSOBSSVI
IEFSSVTI TYPE=FINAL
*     END INITRTN

** TYPE=LIST parameter of IEFSSVTI **

The IEFSSVTI macro with the LIST parameter produces a DSECT that maps the
format of the function routine input table.

**Syntax**

The syntax of the IEFSSVTI macro with TYPE=LIST is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSVTI.</td>
</tr>
<tr>
<td>IEFSSVTI</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFSSVTI.</td>
</tr>
</tbody>
</table>
IEFSSVTI macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE=LIST</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

**TYPE=LIST**

A parameter that defines a DSECT that maps the format of the function routine input table. A TYPE=LIST request is required if a TYPE=SET request or TYPE=COPY request is used in the calling program.

**TYPE=INITIAL** parameter of IEFSSVTI

The IEFSSVTI macro with the INITIAL parameter begins the definition of a static function routine input table.

**Syntax**

The syntax of the IEFSSVTI macro with TYPE=INITIAL is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSVTI.</td>
</tr>
<tr>
<td>IEFSSVTI</td>
<td></td>
</tr>
<tr>
<td>TYPE=INITIAL</td>
<td></td>
</tr>
<tr>
<td>,SSVTDATA=ssvtdata</td>
<td>ssvtdata: RS-type address</td>
</tr>
<tr>
<td>,TABLEN=tablen</td>
<td>ssvtdata: RS-type address</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

**TYPE=INITIAL**

A parameter that begins a static function routine input table build request. This TYPE=INITIAL request is the first request required to build the static function routine input table. The order in which you invoke requests to build the static function routine input table follows:

- TYPE=INITIAL
IEFSSVTI macro

- TYPE=ENTRY
- TYPE=FINAL

,SSVTDATA=ssvtdata
A required parameter that specifies the name of the function routine input table that you are building that relates supported SSVT function codes with function routines. Use this name when referencing the function routine input table and also on the SSVTDATA parameter of the IEFSSVT macro.

,TABLEN=tablen
A required parameter that specifies the name of a constant that the IEFSSVTI macro generates to define the length of the storage required by the function routine input table.

Use this parameter with a TYPE=RESERVE request to reserve dynamic storage when copying the function routine input table for TYPE=SET request modifications.

**TYPE=ENTRY parameter of IEFSSVTI**

The IEFSSVTI macro with the ENTRY parameter defines a function routine entry in a static input table.

**Syntax**

The syntax of the IEFSSVTI macro with TYPE=ENTRY is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSVTI.</td>
</tr>
<tr>
<td>IEFSSVTI</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFSSVTI.</td>
</tr>
<tr>
<td>TYPE=ENTRY</td>
<td></td>
</tr>
<tr>
<td>,FUNCNAME=funcname</td>
<td><code>funcname</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,FUNCADDR=funcaddr</td>
<td><code>funcaddr</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td><strong>Default</strong>: none</td>
</tr>
<tr>
<td>,NUMFCODES=numfcodes</td>
<td><code>numfcodes</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,FCODES=fcodes(,fcodes...)</td>
<td><code>fcodes</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,FCODES=0</td>
<td><strong>Default</strong>: FCODES=0</td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

**TYPE=ENTRY**
A parameter that defines a function routine entry in a static input table. A static input table must contain at least one TYPE=ENTRY invocation. You must specify a TYPE=INITIAL request prior to specifying this TYPE=ENTRY request.

**FUNCNAME=funcname**
An optional set of parameters. You can specify only one of the following:

**FUNCNAME=funcname**
The function routine name. This name can be no more than 8 characters long, beginning with an alphabetic character or national (#, @, or $) character. The remaining characters can be alphabetic, national or numeric. This field must be left-justified and padded to the right with blanks.

For a TYPE=ENTRY request, if you omit FUNCNAME, you must provide the function routine information about a subsequent TYPE=SET request.

**FUNCADDR=funcaddr**
A field that contains the address of the function routine. Specifying FUNCADDR on a TYPE=ENTRY request, reserves storage in the function routine input table for the function routine address.

You must provide the actual address in a subsequent TYPE=SET request.

When you specify FUNCADDR on a TYPE=SET request, you can use the high-order bit to specify the function routine AMODE. Setting this bit indicates that the routine receives control in AMODE 31. Clearing this bit indicates that the routine receives control in AMODE 24. You can also use the FUCNAMODE key to indicate the AMODE of a function routine.

**NUMFCODES=numfcodes**
A required 2-byte parameter that defines the number of function codes supported by the associated function routine. This input field contains a decimal value that must be in the range of 1 to 255.

Use the NUMFCODES parameter to reserve storage for function code information in a static function routine input table entry. You must specify a number greater or equal to the number of function codes specified with the FCODES parameter.

If you do not know the actual number of function codes associated with the routine, specify the maximum number of function codes you expect, to reserve enough storage. In this case, the FCODES parameter of a subsequent TYPE=SET request provides the actual function code information.

**FCODES=fcodes**, **FCODES=0**
An optional parameter that specifies the function codes supported by the associated function routine. This input field may contain either a value or a list of values that must be in the range of 1 to 255. The values do not have to be numbers, they can also be assembler equates.

The same function code value cannot appear more than once within a set of IEFSSVTI invocations representing a function routine input table.
IEFSSVTI macro

For a TYPE=ENTRY request, if you do not specify the FCODES parameter, you must provide the supported function codes on a subsequent TYPE=SET request. The default is 0.

**TYPE=FINAL parameter of IEFSSVTI**

The IEFSSVTI macro with the FINAL parameter ends the definition of a static function routine input table.

**Syntax**

The syntax of the IEFSSVTI macro with TYPE=FINAL is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSVTI.</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

**TYPE=FINAL**

A parameter that ends a static function routine input table build request. This TYPE=FINAL request is the last request required to build the static function routine input table.

**TYPE=SET parameter of IEFSSVTI**

The IEFSSVTI macro with the SET parameter modifies a function routine entry in an existing input table.

**Syntax**

The syntax of the IEFSSVTI macro with TYPE=SET is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSVTI.</td>
</tr>
<tr>
<td>IEFSSVTI</td>
<td></td>
</tr>
</tbody>
</table>
IEFSSVTI macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>b</code></td>
<td>One or more blanks must follow IEFSSVTI.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TYPE=SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>,SSVTDATA=ssvtdata</td>
</tr>
<tr>
<td>,SOURCE=source</td>
</tr>
<tr>
<td>,ENTRYDATA=entrydata</td>
</tr>
</tbody>
</table>

| ,FUNCNAME=funcname | funcname: RS-type address or address in register (2) - (12). |
| ,FUNCADDR=funcaddr | funcaddr: RS-type address or address in register (2) - (12). |

| Default: none |

| ,FUNCAMODE=HOB | Default: FUNCAMODE=HOB |
| ,FUNCAMODE=31 |
| ,FUNCAMODE=24 |

| ,FCODES=fcodes(fcodes...) | fcodes: RS-type address or address in register (2) - (12). |
| ,FCODES=0 | Default: FCODES=0 |

Parameters

The parameters are explained as follows:

**TYPE=SET**
A parameter that modifies a function routine entry in an existing input table. You can use TYPE=SET to modify either a table in dynamic storage, or a static table in non-reentrant programs.

**,SSVTDATA=ssvtdata**
A required parameter that specifies the name of the function routine input table to be modified. The name must match the name of a table specified on the SSVTDATA parameter of the TYPE=INITIAL or TYPE=RESERVE IEFSSVTI macro invocation.

**,SOURCE=source**
A required parameter that specifies the name of the original function routine input table from which the table to be modified was copied.

The name must match the name of a function routine input table that you specified in the SSVTDATA parameter on a TYPE=INITIAL invocation or a TYPE=RESERVE invocation. This information is used with the ENTRYDATA parameter to calculate the offset of the function routine input table you want to modify.
IEFSSVTI macro

The SOURCE parameter and SSVTDATA parameter can refer to the same function routine input table. For example, you may want to modify a static function routine input table that was created by a set of TYPE=INITIAL and TYPE=FINAL invocations, which can be done only in non-reentrant modules.

\textbf{ENTRYDATA=entrydata}

A required 4-byte parameter that specifies the name (or address) of the index of the function routine input table entry that you want to modify. This input field may be either a constant, an assembler equate, or decimal value.

If the value of the ENTRYDATA parameter is greater than the number of function routines in the source table, the target function routine input table does not change. If this is the case, you will not receive a compile-time warning message, because this situation is determined at run time.

\textbf{FUNCNAME=funcname, FUNCADDR=funcaddr}

An optional set of parameters. You can specify only one of the following:

\textbf{FUNCNAME=funcname}

The function routine name. This name can be no more than 8 characters long, beginning with an alphabetic character or national (#, @, or, $) character. The remaining characters can be alphabetic, national, or numeric.

This field must be left-justified and padded to the right with blanks.

For a TYPE=ENTRY request, if you omit FUNCNAME, you must provide the function routine information about this request.

\textbf{FUNCADDR=funcaddr}

The pointer input that contains the address of the function routine. Specifying FUNCADDR on a TYPE=ENTRY invocation, reserves storage in the function routine input table for the function routine address.

If you specify FUNCADDR on this invocation, you can use the high-order bit to specify the function routine AMODE. Setting this bit indicates that the routine receives control in AMODE 31. Clearing this bit indicates that the routine receives control in AMODE 24. You can also use the FUNCAMODE key to indicate the AMODE of a function routine.

\textbf{FUNCAMODE=HOB, FUNCAMODE=31, FUNCAMODE=24}

An optional input parameter that specifies the AMODE of a function routine identified by the address. The SSI uses this information to determine the AMODE in which the function routine receives control.

If you do not specify the FUNCAMODE parameter, the high-order bit of the address specified with the FUNCADDR parameter indicates the AMODE of the function routine. If the high-order bit is on, the function routine is treated as AMODE 31.

FUNCAMODE=HOB specifies that the high-order bit of a function routine address indicates the AMODE in which the function routine receives control. HOB is the default.

FUNCAMODE=31 specifies that the function routine receives control in AMODE 31.

FUNCAMODE=24 specifies that the function routine receives control in AMODE 24.

\textbf{FCODES=fcodes(,fcodes...)}
IEFSSVTI macro

,FCODES=0

An optional byte parameter that specifies the function codes supported by the associated function routine. This input field contains a decimal value that must be in the range of 1 to 255.

The same function code value cannot appear more than once within a set of IEFSSVTI invocations representing a function routine input table.

You must specify the FCODES parameter if the function code information was not provided on the TYPE=ENTRY invocation that corresponds to the entry being modified.

Function codes that you specify with the TYPE=SET invocation replace any function codes specified on the original TYPE=ENTRY invocation. If you do not specify the FCODES parameter the function code information in the entry being modified is unchanged.

If you specify more function codes with the FCODES parameter than the maximum number of function codes for which room was reserved in the table entry being modified, the function code information in the target entry is unchanged. The IEFSSVTI macro does not provide a warning.

The default is 0.

**TYPE=RESERVE parameter of IEFSSVTI**

The IEFSSVTI macro with the RESERVE parameter reserves dynamic storage for a function routine input table.

**Syntax**

The syntax of the IEFSSVTI macro with TYPE=RESERVE is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSVTI.</td>
</tr>
<tr>
<td>IEFSSVTI</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFSSVTI.</td>
</tr>
<tr>
<td>TYPE=RESERVE</td>
<td></td>
</tr>
<tr>
<td>,SSVTDATA=ssvtdata</td>
<td><code>ssvtdata</code>: RS-type address</td>
</tr>
<tr>
<td>,TABLEN=tablen</td>
<td><code>tablen</code>: RS-type address</td>
</tr>
<tr>
<td>,MAXFCODES=maxcodes</td>
<td><code>maxcodes</code>: RS-type address or address in register (2) - (12).</td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

**TYPE=RESERVE**
A parameter that reserves the amount of dynamic storage required to contain a copy of a static function routine input table.

,**SSVTDATA=ssvtdata**
A required parameter that specifies the name of the function routine input table that relates supported SSVT function codes with function routines. Use this name when referencing the function routine input table and also on the SSVTDATA parameter of the IEFSSVT macro.

,**TABLEN=tablen**
,**MAXFCODES=maxfcodes**
You must specify one of the following parameters:

**TABLEN=tablen**
A parameter that specifies the name of a constant, which contains the length of storage required by the function routine input table. This should be the name of a constant specified by the TABLEN parameter on a previous TYPE=INITIAL invocation. Use this parameter when reserving storage for a dynamic function routine input table that is to be copied from another table.

**MAXFCODES=maxfcodes**
A parameter that specifies the maximum number of function codes that is supported by the entire function routine input table. Use this key to reserve space for a dynamic input table when the specific function routines and function codes that are supported are not known at compile time.

**TYPE=COPY parameter of IEFSSVTI**

The IEFSSVTI macro with the COPY parameter copies a static function routine input table to dynamic storage.

**Syntax**

The syntax of the IEFSSVTI macro with TYPE=COPY is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IEFSSVTI.</td>
</tr>
<tr>
<td>IEFSSVTI</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IEFSSVTI.</td>
</tr>
<tr>
<td>TYPE=COPY</td>
<td></td>
</tr>
<tr>
<td>,SSVTDATA=ssvtdata</td>
<td>ssvtdata: RS-type address</td>
</tr>
</tbody>
</table>
IEFSSVTI macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE=source</td>
<td>source: RS-type address or address in register (2) - (12).</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

**TYPE=COPY**

A parameter copies a static function routine input table to dynamic storage.

The TYPE=COPY invocation expands inline to copy the table identified by the SOURCE parameter to the table identified by the SSVTDATA parameter. The source table contains the information for the macro to calculate the length that needs to be moved.

**SSVTDATA=ssvtdata**

A required parameter that specifies the name of the target function routine input table (the destination for the copy). This name must match the name of a table specified by the SSVTDATA parameter on a TYPE=RESERVE invocation.

**SOURCE=source**

A required parameter that specifies the name of the table to be copied. The name must match the name of a table that was specified on the SSVTDATA parameter on an TYPE=INITIAL invocation.
Chapter 83. IFAQUERY — SMF configuration query service

Description
The IFAQUERY service provides SMF configuration information to its caller. The IFAQUERY service currently performs the following function:

- RETRIEVE STATUS - Return the status of SMF recording. Information about SMF LOGSTREAMs is returned, including the SMF record types being written to the log stream. When SMF is not recording, a non-zero return code is returned.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state and system PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task mode</td>
</tr>
<tr>
<td>Cross Memory Mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements
None.

Restrictions
None.

Input register information
There are no input register requirements for issuing the IFAQUERY macro.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
### IFAQUERY service

- **2-13**  Unchanged
- **14-15**  Used as work registers by the system

#### Performance implications
None.

#### Syntax
The standard form of the IFAQUERY macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede IFAQUERY.</td>
</tr>
<tr>
<td>IFAQUERY</td>
<td>One or more blanks must follow IFAQUERY.</td>
</tr>
<tr>
<td><code>[xlabel]</code></td>
<td>An optional symbol, starting in column 1, that is the name on the IFAQUERY macro invocation. DEFAULT: No name</td>
</tr>
<tr>
<td>RETRIEVE</td>
<td></td>
</tr>
<tr>
<td><code>_STATUS</code></td>
<td></td>
</tr>
<tr>
<td><code>,OUTAREA=ouarea</code></td>
<td><code>ouarea</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,OUTLEN=oulen</code></td>
<td><code>oulen</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>[DETAILS=LOGSTREAM]</code></td>
<td>Default: ALL</td>
</tr>
<tr>
<td><code>[RETCODE=retcode]</code></td>
<td><code>retcode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>[RSNCODE=rsncode]</code></td>
<td><code>rsncode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>`[PLISTVER=plistver</td>
<td>IMPLIED_VERSION]`</td>
</tr>
<tr>
<td><code>[MF=S]</code></td>
<td>Default: MF=S</td>
</tr>
<tr>
<td><code>[MF=(L,mfctrl,mfattr, 0D)]</code></td>
<td></td>
</tr>
<tr>
<td><code>[MF=(E,mfctrl,COMPLETE)]</code></td>
<td></td>
</tr>
</tbody>
</table>
Parameters

In the following set of mutually exclusive keywords, only one keyword must be specified.

**RETRIEVE**
Retrieve SMF Recording information.

**STATUS**
Obtain information about the current log stream recording environment.

**OUTAREA=** `outarea`
A required character input/output specifying an area to contain the data being returned by IFAQUERY. The answer area is defined by the IFAQUAA macro, which consists of the QUaHDR and QUALSI structures. The IFAQUAA mapping macro provides the format of the area. The area can be in the primary address space or in an address space or data space that is addressable through a public entry on the caller's DU-AL. Use the **OUTLEN** parameter to specify the length of the area.

To code: Specify the RS-type address of a character field, or register (2) - (12) (ASM only).

**OUTLEN=** `outlen`
A required fullword input parameter that contains the length of the area provided to contain the data being returned by IFAQUERY.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) (ASM only).

End of the mutually exclusive keywords.

**DETAILS=LOGSTREAM**
An optional keyword input indicating the type of information that the SMF query service should return.

DEFAULT: LOGSTREAM.

**RETCODE=** `retcode`
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) (ASM only).

**RSNCODE=** `rsncode`
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or address in register (2) - (12) (ASM only).

**PLISTVER=** `plistver` | **IMPLIED_VERSION**
An optional byte input decimal value in the "0-0" range that specifies the macro version. PLISTVER is the only parameter allowed on the list form of MF. This parameter determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values can be:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
**IFAQUERY service**

- **MAX**, if you want the parameter list to be the largest size currently supported. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM suggests that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **1**, if you use the currently available parameters.

**To code:** Specify one of the following:

- IMPLIED_VERSION
- MAX
- 1

, MF=S
, MF=(L, mfctrl, mfattr, 0D)

MF=(E, fctrl, COMPLETE)

An optional keyword input that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

, fctrl

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1) - (12).

mfattr

An optional 1-60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

COMPLETE

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

**ABEND codes**

None.

**Return and reason codes**

The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning for each return and reason code.
### Table 77. Return and Reason Codes for the IFAQUERY Macro

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None</td>
<td><strong>Explanation:</strong> IFAQUERY request successful.</td>
</tr>
<tr>
<td>04</td>
<td></td>
<td><strong>Explanation:</strong> Warning. Refer to the action provided with the specific reason code.</td>
</tr>
</tbody>
</table>
| 04          | xxxxxxx01   | **Explanation:** OUTAREA is too small to contain all the requested data. The results in the OUTAREA were truncated. QUAHLEN specifies the amount of storage that is required to return a complete result.  
**Action:** Try the request again with a larger OUTAREA. |
| 04          | xxxxxxx03   | **Explanation:** DETAILS=LOGSTREAM was requested but no log stream information exists. The QUAHDR is filled in appropriately. No records are returned.  
**Action:** None. |
| 08          |             | **Explanation:** Incorrect input parameter. Refer to the action provided with the specific reason code. |
| 08          | xxxxxxx01   | **Explanation:** Caller was not running as a task.  
**Action:** Move the invocation of IFAQUERY under a task. |
| 08          | xxxxxxx02   | **Explanation:** The input parmlist cannot be accessed.  
**Action:** Check for one of the following possible errors:  
- Program exception during access of parameter list.  
- Parameter list has incorrect address. |
| 08          | xxxxxxx03   | **Explanation:** The QUA area could not be accessed.  
**Action:** Check for one of the following possible errors:  
- Program exception during access of QUA area.  
- QUA area has incorrect address. |
| 08          | xxxxxxx05   | **Explanation:** The OUTAREA length is too small for a QUA header.  
**Action:** Increase the size of the OUTAREA. The length must be greater than or equal to 16 bytes. |
| 08          | xxxxxxx06   | **Explanation:** QUA has invalid ALET. |
| 0C          |             | **Explanation:** Environmental error. Refer to the action provided with the specific reason code. |
| 0C          | xxxxxxx01   | **Explanation:** SMF recording is not active. No records are returned.  
**Action:** None. |
| 0C          | xxxxxxx02   | **Explanation:** Storage for local area was not obtained.  
**Action:** None. |
| 10          | None        | **Explanation:** Unexpected error. The state of the request is unpredictable. |
IFQUERY service
Chapter 84. IOCINFO — Obtain MVS I/O configuration information

Description

Use the IOCINFO macro to obtain the following I/O configuration information:

- I/O configuration token
- Default channel subsystem identifier for the logical partition
- The maximum device measurement block index that is currently assigned
- The I/O facilities that are supported and enabled by the hardware and software.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state, with any PSW key.</td>
</tr>
<tr>
<td></td>
<td>For LINKAGE=BRANCH, all of the following:</td>
</tr>
<tr>
<td></td>
<td>• Supervisor state with PSW key 0</td>
</tr>
<tr>
<td></td>
<td>• 31-bit addressing mode</td>
</tr>
<tr>
<td></td>
<td>• Primary ASC mode</td>
</tr>
<tr>
<td></td>
<td>• Parameter list and any data areas it points to must be in fixed storage or, if the caller is disabled, in disabled reference (DREF) storage</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24- or 31- bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt Status:</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller may hold locks, but is not required to hold any</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space or be in an address/data space that is addressable through a public entry on the caller’s dispatchable unit access list (DU-AL).</td>
</tr>
</tbody>
</table>

Programming requirements

If in AR mode, specify SYSSTATE ASCENV=AR before invoking the macro.

Restrictions

None.

Input register information

Before issuing the IOCINFO macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:
IOCINFO macro

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2-13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
</tr>
<tr>
<td>2-13</td>
</tr>
<tr>
<td>14-15</td>
</tr>
</tbody>
</table>

Performance implications

None.

Syntax

The standard form of the IOCINFO macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOCINFO.</td>
</tr>
</tbody>
</table>

IOCINFO

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>One or more blanks must follow IOCINFO.</td>
</tr>
</tbody>
</table>

IOCTOKEN=ioctoken addr | ioctoken addr: RX-type address or register (2) - (12). |

,DCMINFO=xdcminfo | xdcminfo: RS-type address or register (2) - (12). |

,CSSID=cssid addr | cssid addr: RX-type address or register (2) - (12). |

,MAXMBI=maxmbi addr | maxmbi addr: RS-type address or register (2) - (12). |

,IOFACILITIES=iofc addr | iofc addr: RX-type address or register (2) - (12). |

,IODFINFO=xiodfinfo | xiodfinfo: RS-type address or register (2) - (12). |
The parameters are explained as follows:

**IOCTOKEN=ioc_token_addr**
- Specifies the address of a 48-character area where the system returns the current MVS I/O configuration token.

**,DCMINFO=xdcminfo**
- Specifies the address of an optional 32 character output area into which IOCINFO is to return Dynamic Channel Path Management (DCM) information which can be mapped by IOSDDCMI.

**,CSSID=cssid_addr**
- Specifies the address of a one byte output area where the system returns the default channel subsystem ID for the logical partition.
  - A return code of X'00', reason code X'00' indicates that the program is running on a processor that supports multiple channel subsystems.
  - A return code of X'00', reason code X'01' indicates that the program is running on a processor that does not support multiple channel subsystems, and the CSS ID assigned is a zero.

**,MAXMBI=maxmbi_addr**
- Specifies the address of a halfword field where the system returns the maximum device measurement block index that is currently assigned.

**,IOFACILITIES=iocf_addr**
- Specifies the address of a required 256-byte output area into which the IOCINFO service returns the I/O facility information. This area is mapped by mapping macro IOSDIOFC.

**,IODFINFO=xiodfinfo**
- Specifies the address of an optional 128 character output area into which IOCINFO is to return IODF information which is mapped by IOSDIODI.

**,LINKAGE=SYSTEM,**
- Specifies the type of call that should be generated:
  - SYSTEM: Specifies a Program Call (PC)
  - BRANCH: Specifies a branch entry
- LINKAGE=BRANCH is intended for performance-sensitive programs.
IOCSIIO macro

**,RETCODE**=retcode addr  
Specifies the fullword location where the system is to store the return code.  
The return code is also in GPR 15.

**,RSNcode**=rsncode addr  
Specifies the fullword location where the system is to store the reason code.  
The reason code is also in GPR 0.

**,PLISTVER**=xplistver  
**,** **PLISTVER**=IMPLIED_VERSION  
An optional byte input decimal value (with a value of 1) that specifies the 
macro version. PLISTVER is the only key allowed on the list form of MF and 
determines which parameter list is generated. Note that MAX may be specified 
instead of a number, and the parameter list will be of the largest size currently 
supported. This size may grow from release to release (thus possibly affecting 
the amount of storage needed by your program). If your program can tolerate 
this, IBM recommends that you always specify MAX when creating the list 
form parameter list as that will ensure that the list form parameter list is 
always long enough to hold whatever parameters might be specified on the 
execute form.

The default is IMPLIED_VERSION. When PLISTVER is omitted, the default is 
the lowest version which allows all of the parameters specified on the 
invocation to be processed.

**ABEND codes**

None.

**Return and reason codes**

When the system returns control to the caller, GPR 15 (and retcode addr, if you 
coded RETCODE) contains the return code. For return code X'08', the reason code 
is in GPR 0 (and rsncode addr, if you coded RSNCODE).

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                      | **Meaning**: Successful completion.  
**Action**: None. |
| 00                      | 01                      | **Meaning**: Successful completion from a CSSID parameter request. The program is running on a  
processor that supports multiple channel subsystems.  
**Action**: None. |
| 08                      | 01                      | **Meaning**: Program error. An ALET in the parameter list is not valid. The caller might have inadvertently written over an area in the parameter list.  
**Action**: Check to see if your program inadvertently overlaid the parameter list generated by the macro. |

207x719]retcode addr
[180x707]Specifies the fullword location where the system is to store the return code.  
The return code is also in GPR 15.

207x719]rsncode addr
[180x666]Specifies the fullword location where the system is to store the reason code.  
The reason code is also in GPR 0.

212x636]xplistver
[162x624],PLISTVER=IMPLIED_VERSION
An optional byte input decimal value (with a value of 1) that specifies the 
macro version. PLISTVER is the only key allowed on the list form of MF and 
determines which parameter list is generated. Note that MAX may be specified 
instead of a number, and the parameter list will be of the largest size currently 
supported. This size may grow from release to release (thus possibly affecting 
the amount of storage needed by your program). If your program can tolerate 
this, IBM recommends that you always specify MAX when creating the list 
form parameter list as that will ensure that the list form parameter list is 
always long enough to hold whatever parameters might be specified on the 
execute form.

The default is IMPLIED_VERSION. When PLISTVER is omitted, the default is 
the lowest version which allows all of the parameters specified on the 
invocation to be processed.

**ABEND codes**

None.

**Return and reason codes**

When the system returns control to the caller, GPR 15 (and retcode addr, if you 
coded RETCODE) contains the return code. For return code X'08', the reason code 
is in GPR 0 (and rsncode addr, if you coded RSNCODE).

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                      | **Meaning**: Successful completion.  
**Action**: None. |
| 00                      | 01                      | **Meaning**: Successful completion from a CSSID parameter request. The program is running on a  
processor that supports multiple channel subsystems.  
**Action**: None. |
| 00                      | 01                      | **Meaning**: Successful completion from a CSSID parameter request. The program is running on a  
processor that does not support multiple channel subsystems and the CSS ID assigned is a zero.  
**Action**: None. |
| 08                      | 01                      | **Meaning**: Program error. An ALET in the parameter list is not valid. The caller might have inadvertently written over an area in the parameter list.  
**Action**: Check to see if your program inadvertently overlaid the parameter list generated by the macro. |
### IOCINFO macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08                      | 02                      | **Meaning**: Program error. The system could not access the caller’s parameter list.  
**Action**: Check to see if your program inadvertently overlaid the parameter list generated by the macro. |
| 08                      | 05                      | **Meaning**: Program error. An error occurred when the system referenced the user-supplied area specified in the IOCTOKEN parameter.  
**Action**: Check to see if your program correctly specified the IOCTOKEN area. |
| 08                      | 09                      | **Meaning**: System error. This reason code is for IBM diagnostic purposes only.  
**Action**: Record the reason code and supply it to the appropriate IBM support personnel. |
| 08                      | 0F                      | **Meaning**: An error occurred referencing the user-supplied area that is specified in the IOFACILITIES parameter.  
**Action**: Check to see if your program correctly specified the IOFACILITIES area. |
| 20                      |                         | **Meaning**: System error. This return code is for IBM diagnostic purposes only.  
**Action**: Record the return code and supply it to the appropriate IBM support personnel. |
| 24                      | 07                      | **Meaning**: Program error. The system does not support the specified parameter.  
**Action**: Check the parameters on the IOCINFO macro to make sure they are valid on your release of the system. |

### IOCINFO—List form

Use the list form of the IOCINFO macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

**Syntax**

The list form of the IOCINFO macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOCINFO.</td>
</tr>
<tr>
<td>IOCINFO</td>
<td></td>
</tr>
</tbody>
</table>
### IOCINFO macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>One or more blanks must follow IOCINFO.</td>
</tr>
<tr>
<td>MF=(L,list addr)</td>
<td>list addr: Symbol.</td>
</tr>
<tr>
<td>MF=(L,list addr,attr)</td>
<td>attr: 1- to 60- character input string</td>
</tr>
<tr>
<td>MF=(L,list addr,0D)</td>
<td>Default: 0D</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the IOCINFO macro with the following exception:

- **MF=(L,list addr)**
- **MF=(L,list addr,attr)**
- **MF=(L,list addr,0D)**

Specifies the list form of the IOCINFO macro.

- **list addr** is the name of a storage area to contain the parameters.

- **attr** is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code **attr**, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

### IOCINFO - Execute form

Use the execute form of the IOCINFO macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

### Syntax

The execute form of the IOCINFO macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOCINFO.</td>
</tr>
<tr>
<td>IOCINFO</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOCINFO.</td>
</tr>
<tr>
<td>IOCTOKEN=ioctoken addr</td>
<td>ioctoken addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_CSSID=cssid addr</td>
<td>cssid addr: RS-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
### IOCINFO macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,MAXMBI=maxmbi addr</td>
<td>maxmbi addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,IOFACILITIES=iofc addr</td>
<td>iofc addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,LINKAGE=SYSTEM</td>
<td><strong>Default</strong>: SYSTEM</td>
</tr>
<tr>
<td>,LINKAGE=BRANCH</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode addr</td>
<td>retcode addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsnconde addr</td>
<td>rsnconde addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td>list addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td>Default: COMPLETE</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the IOCINFO macro with the following exceptions:

- ,MF=(E,list addr)
- ,MF=(E,list addr,COMPLETE)

Specifies the execute form of the IOCINFO macro.

- **list addr** specifies the area that the system uses to contain the parameters.
- **COMPLETE**, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.
IOCINFO macro
Chapter 85. IOSADMF — Transfer hiperspace data

Description

The IOSADMF macro provides an interface for the movement of large amounts of data between main and expanded storage.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or program key mask (PKM) allowing keys 0-7.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB mode for AREAD, AWRITE, and AQUERY requests.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

The caller’s parameter list and range list must be in the primary address space.

Restrictions

For IOSADMF APURGE requests, the caller may not have an EUT FRR established.

Input register information

Before issuing the IOSADMF macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>
IOSADMF macro

2-13  Unchanged
14-15  Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
Using IOSADMF to move large amounts of data between central and expanded storage is more efficient than synchronous methods of moving data.

Syntax
The standard form of the IOSADMF macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSADMF.</td>
</tr>
<tr>
<td>IOSADMF</td>
<td>One or more blanks must follow IOSADMF.</td>
</tr>
<tr>
<td>AREAD</td>
<td></td>
</tr>
<tr>
<td>AWRITE</td>
<td></td>
</tr>
<tr>
<td>AQUERY</td>
<td></td>
</tr>
<tr>
<td>,ALET=alet-addr</td>
<td>alet-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,NUMRANGE=n</td>
<td>n: Number from 1 to 125.</td>
</tr>
<tr>
<td>,NUMRANGE=num-addr</td>
<td>num-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td>Default: NUMRANGE=1.</td>
</tr>
<tr>
<td>,RANGLIST=list-addr</td>
<td>list-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,FAILBLKP=fail-addr</td>
<td>fail-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,CROSSOVER=cross-addr</td>
<td>cross-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RETCODE=ret-addr</td>
<td>ret-addr: RX-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;RSNCODE=rsn-addr</td>
<td>rsn-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>&amp;MF=S</td>
<td></td>
</tr>
</tbody>
</table>

Table 78. Parameters Valid with IOSADMF Requests

<table>
<thead>
<tr>
<th>Parameters</th>
<th>APURGE</th>
<th>AREAD</th>
<th>AWRITE</th>
<th>AQUERY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALET</td>
<td>required</td>
<td>required</td>
<td>required</td>
<td>not valid</td>
</tr>
<tr>
<td>NUMRANGE</td>
<td>not valid</td>
<td>optional</td>
<td>optional</td>
<td>not valid</td>
</tr>
<tr>
<td>RANGLIST</td>
<td>not valid</td>
<td>required</td>
<td>required</td>
<td>not valid</td>
</tr>
<tr>
<td>FAILBLKP</td>
<td>not valid</td>
<td>optional</td>
<td>optional</td>
<td>not valid</td>
</tr>
<tr>
<td>CROSSOVER</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>optional</td>
</tr>
<tr>
<td>RETCODE</td>
<td>optional</td>
<td>optional</td>
<td>optional</td>
<td>optional</td>
</tr>
<tr>
<td>RSNCODE</td>
<td>optional</td>
<td>optional</td>
<td>optional</td>
<td>optional</td>
</tr>
<tr>
<td>MF</td>
<td>optional</td>
<td>optional</td>
<td>optional</td>
<td>optional</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

**APURGE**

Specifies the type of request, as follows: APURGE requests that the system purge any active AREAD or AWRITE operation for the hiperspace represented by the ALET.

**AREAD**

Requests that the system transfer data from a hiperspace to an address space.

**AWRITE**

Requests that the system transfer data from an address space to a hiperspace.

**AQUERY**

Requests that the system check to determine whether ADMF (asynchronous data mover facility) is installed. If ADMF is installed, the system returns a return code of 0. If ADMF is not installed, the system returns a return code of 8 with a corresponding reason code.

**ALET=alet-addr**

Specifies either the address of a fullword or a register that contains the ALET associated with the hiperspace that is the target of an APURGE, AREAD, or AWRITE request.

**NUMRANGE=n**

Specifies the number of entries in the range list in one of the following formats:

- A decimal digit from 1 through 125
- A fullword that contains the number of entries
- A register that contains the address of a fullword that contains the number of entries.
The default is NUMRANGE=1.

,RANGLIST=list-addr
Specifies a fullword that contains the address of a list of ranges (up to 125), or specifies a register that contains the address of the fullword pointer to the range list. The list of ranges specifies one or more virtual storage ranges that are to be moved. The range list consists of a number of entries (specified by NUMRANGE), where each entry consists of three words:

First word
The starting virtual address in the address space into which the data is to be read or from which data is to be written.

Second word
The starting virtual address in the hiperspace from which the system is to read or into which the system is to read.

Third word
The number of blocks the system is to read from the hiperspace or write from the address space.

For example, if your register and storage are set up as in Figure 9, you can code the RANGLIST parameter and NUMRANGE parameter as follows:

NUMRANGE=3 ,RANGLIST=(5)

or

NUMRANGE=3, RANGLIST=RANGADDR

Figure 9. RANGLIST and NUMRANGE Parameters

The range list must be in the caller's primary address space.

,FAILBLKP=fail-addr
Specifies a fullword that contains the address of a range list entry, or specifies a register that contains the address of the fullword pointer to a range list entry, for which a failure occurred. The system returns this value only when you code FAILBLKP and when the system can identify the failing range list entry.

When the system returns a return code 8 and fail-addr contains a non-zero value, the entry identified by fail-addr is either partially processed, or not processed and any subsequent range list entries are not processed. However, any prior range list entries processed successfully.

fail-addr contains a non-zero value only when the failing range list is known. The reason codes indicate when fail-addr is set.

,CROSSOVER=cross-addr
Specifies a fullword or register in which the system is to place the
system-implemented crossover value. If the number of pages requested to be moved is greater than the CROSSOVER value, the system moves the data asynchronously with the ADMF. If you invoke IOSADMF when the number of pages is less than the crossover value, the system uses the move page facility to move the data.

You can request this value to determine whether using the ADMF is warranted for particular data movement.

,RETCODE=ret-addr
   Specifies the location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=rsn-addr
   Specifies the location where the system is to store the return code. The reason code is also in GPR 0.

,MF=S
   Specifies the standard form of the macro. This form generates code to place the parameters into an inline parameter list and invoke the macro service.

### ABEND codes

None.

### Return and reason codes

When the IOSADMF macro returns control to the caller, GPR 15 (and ret-addr, if you coded RETCODE) contains a return code and GPR 0 (and rsn-addr, if you coded RSNCODE) contains a reason code.

The reason code consists of four bytes; the third byte contains a value that indicates where the error occurred. The third byte contains X'01' when the error occurred in an address space; it contains X'02' when the error occurred in a hiperspace.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | —                       | **Meaning:** The IOSADMF operation completed successfully. For an AQUERY request, return code 0 indicates that the ADMF is installed.  
**Action:** None. |
| 04                      | xx0101xx                | **Meaning:** System error. The IOSADMF operation failed because of a communication error. The request was started, but the system stopped the request because of an error condition.  
The failure occurred in the storage area whose address is in the first word of the input range list entry. FAILBLKP contains the address of the range list entry for which the failure occurred.  
**Action:** Either retry the operation using IOSADMF or use the HSPSERV macro. If you still get the same error, record the return and reason codes; contact hardware support. |

Table 79. Return and Reason Codes for the IOSADMF Macro
### Table 79. Return and Reason Codes for the IOSADMF Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>xx0202xx</td>
<td><strong>Meaning:</strong> System error. The IOSADMF operation failed because of a communication error. The request was started, but the system stopped the request because of an error condition. The failure occurred in the storage whose address is in the second word of the input range list entry. FAILBLKP contains the address of the range list entry for which the failure occurred. <strong>Action:</strong> Either retry the operation using IOSADMF or use the HSPSERV macro. If you still get the same error, record the return and reason codes; contact hardware support.</td>
</tr>
<tr>
<td>04</td>
<td>xx0301xx</td>
<td><strong>Meaning:</strong> Program error. A specified address identified an area in an address space for which the caller is not authorized. A protection exception was encountered. The failure occurred in the storage area whose address is in the first word of the input range list entry. FAILBLKP contains the address range list entry for which the failure occurred. <strong>Action:</strong> Either specify the address of an address space that the user has the authority to access, or obtain adequate authority to use the specified address. Retry the operation using IOSADMF or use the HSPSERV macro.</td>
</tr>
<tr>
<td>04</td>
<td>xx0501xx</td>
<td><strong>Meaning:</strong> Program error. An error occurred during address translation. The request cannot be completed at the current time because an address space page was not valid. Either the address in the first word of the input range list entry was not correct or identified an area that was not backed. The failure occurred in the storage area whose address is in the first word of the input range list entry. FAILBLKP contains the range list entry for which the failure occurred. <strong>Action:</strong> Either retry the operation using IOSADMF or use the HSPSERV macro. Ensure that all the pages that are to be used are page fixed.</td>
</tr>
<tr>
<td>04</td>
<td>xx0502xx</td>
<td><strong>Meaning:</strong> Program error. An error occurred during address translation. The request cannot be completed at the current time because a hiperspace page was not valid. Either the hiperspace in the second word of the input range list entry was not correct or identified an area that was reclaimed by the system. The failure occurred in the storage area whose address is in the second word of the input range list entry. FAILBLKP contains the address of a range list entry for which the failure occurred. <strong>Action:</strong> Use the HSPSERV macro to restore the hiperspace page.</td>
</tr>
</tbody>
</table>
## Table 79. Return and Reason Codes for the IOSADMF Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04                      | xx0601xx                | **Meaning:** System error. An uncorrectable storage error occurred at either the source or destination of the data move.  
The failure occurred in the storage area whose address is in the first word of the input range list entry. FAILBLKP contains the range list entry for which the failure occurred.  
**Action:** Either retry the operation using IOSADMF or use the HSPSERV macro. If you still get the same error, record the return and reason codes; contact hardware support. |
| 04                      | xx0702xx                | **Meaning:** System error. An uncorrectable storage error occurred at either the source or destination of the data move.  
The failure occurred in the storage area whose address is in the second word of the input range list entry. FAILBLKP contains the range list entry for which the failure occurred.  
**Action:** Either retry the operation using IOSADMF or use the HSPSERV macro. If you still get the same error, record the return and reason codes; contact hardware support. |
| 04                      | xx0Cxxxxx               | **Meaning:** System error. An uncorrectable storage error occurred at either the source or destination of the data move.  
The system could not determine whether the error occurred in the address space storage or the hiperspace storage. FAILBLKP contains the range list entry for which the failure occurred.  
**Action:** Either retry the operation using IOSADMF or use the HSPSERV macro. If you still get the same error, record the return and reason codes; contact hardware support. |
| 08                      | xx31xxxxx               | **Meaning:** Environmental error. The ADMF is not installed on the current system. The ADMF cannot be used until both hardware and software are installed and the operating system is IPLed.  
**Action:** Retry the operation using the HSPSERV macro instead of the IOSADMF macro. |
| 08                      | xx32xxxxx               | **Meaning:** System error. The asynchronous data mover facility is not available. The system detected an unrecoverable error.  
**Action:** Use the HSPSERV macro instead of IOSADMF and rerun the program. Record the return and reason codes; contact hardware support. |
| 08                      | xx34xxxxx               | **Meaning:** Program error. The calling program does not meet one or more of the environmental requirements for using IOSADMF.  
**Action:** Ensure that the IOSADMF macro is issued in the required environment. See in "Environment" on page 909. |
| 08                      | xx35xxx                 | **Meaning:** Program error. Either no option (AWRITE, AREAD, APURGE, or AQUERY) was specified on the IOSADMF macro, or more than one option was specified. This error can occur if the parameter list is overlaid.  
**Action:** Make sure the IOSADMF macro invocation specifies one option and rerun the program. |
### Table 79. Return and Reason Codes for the IOSADMF Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08                      | xx36xxxx                | **Meaning:** Program error. The specified ALET is incorrect. The ALET did not designate a hiperspace, or the ALET is not on the caller's access list.  
**Action:** Make sure the ALET is valid and rerun the program. |
| 08                      | xx37xxxx                | **Meaning:** Program error. The range count is not valid. The NUMRANGE value specified is either less than 1 or greater than 125.  
**Action:** Specify a NUMRANGE value from 1 through 125 and rerun the program. |
| 08                      | xx38xxxx                | **Meaning:** Program error. An input parameter list could not be addressed, or an error occurred during a reference to a range list entry. The RANGLIST parameter may be specified incorrectly.  
**Action:** Ensure RANGLIST is specified correctly, and NUMRANGE is a valid value, and rerun the program. |
| 08                      | xx39xxxx                | **Meaning:** Program error. An error occurred during the processing of a RANGLIST entry address. FAILBLKP contains the address of the failing entry.  
**Action:** Ensure the following:  
• The RANGLIST parameter is correctly specified  
• The address on the RANGLIST parameter is correct  
• The NUMRANGE value reflects the actual number of NUMRANGE entries.  
• The NUMRANGE value is from 1 though 125.  
Rerun the program. |
| 08                      | xx3Axxxx                | **Meaning:** System error. This return and reason code combination is for IBM diagnostic purposes only.  
**Action:** Record the return and reason codes and supply them to the IBM Support Center. |
| 08                      | xx3Bxxxx                | **Meaning:** Program error. The calling program does not meet one or more of the environmental requirements for using IOSADMF.  
**Action:** Ensure IOSADMF is issued in the required environment. See in "Environment" on page 909. |
| 08                      | xx3Cxxxx                | **Meaning:** Program error. An incorrect version of the ADMF was specified. The current version is 1. This error can occur if the parameter list is overlaid.  
**Action:** Contact your software support. |
| 08                      | xx3Exxxx                | **Meaning:** Program error. The reserved fields in XFLAGS, XRESERVED1, or XRESERVED2 are not zero. These fields must be set to zero before the IOSADMF macro can be invoked.  
**Action:** See the IOSADMF macro expansion. Correct the parameter list and rerun the program. |
### IOSADMF - List form

Use the list form of the IOSADMF macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

#### Syntax

The list form of the IOSADMF macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede IOSADMF.</td>
</tr>
<tr>
<td>IOSADMF</td>
<td><code>list addr</code>: symbol.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must follow IOSADMF.</td>
</tr>
<tr>
<td><code>,MF=(L,list addr)</code></td>
<td><code>list addr</code>: symbol.</td>
</tr>
<tr>
<td><code>,MF=(L,list addr,attr)</code></td>
<td><code>attr</code>: 1- to 60-character input string.</td>
</tr>
<tr>
<td><code>,MF=(L,list addr,0D)</code></td>
<td><code>Default</code>: 0D</td>
</tr>
</tbody>
</table>
IOSADMF macro

### Parameters

The parameters are explained under the standard form of the SAMPLE macro with the following exception:

- **MF=(L,list addr)**
- **MF=(L,list addr,attr)**
- **MF=(L,list addr,0D)**

Specifies the list form of the IOSADMF macro.

*list addr* is the name of a storage area to contain the parameters.

*attr* is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

### IOSADMF - Execute form

Use the execute form of the IOSADMF macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

### Syntax

The execute form of the IOSADMF macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td><strong>b</strong></td>
<td>One or more blanks must precede IOSADMF.</td>
</tr>
<tr>
<td>IOSADMF</td>
<td></td>
</tr>
<tr>
<td><strong>b</strong></td>
<td>One or more blanks must follow IOSADMF.</td>
</tr>
<tr>
<td>APURGE</td>
<td></td>
</tr>
<tr>
<td>AREAD</td>
<td></td>
</tr>
<tr>
<td>AWRITE</td>
<td></td>
</tr>
<tr>
<td>AQURY</td>
<td></td>
</tr>
<tr>
<td>,ALET=alet-addr</td>
<td><em>alet-addr</em>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,NUMRANGE=n</td>
<td><em>n</em>: Decimal digit from 1 to 125.</td>
</tr>
<tr>
<td>,NUMRANGE=num-addr</td>
<td><em>num-addr</em>: RX-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default: NUMRANGE=1.</td>
<td></td>
</tr>
<tr>
<td><strong>,RANGLIST=list-addr</strong></td>
<td>list-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><strong>,FAILBLKP=fail-addr</strong></td>
<td>fail-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><strong>,CROSSOVER=cross-addr</strong></td>
<td>cross-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><strong>,RETCODE=ret-addr</strong></td>
<td>ret-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><strong>,RSNCODE=rsn-addr</strong></td>
<td>rsn-addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><strong>,MF=(E,list addr)</strong></td>
<td>list addr: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td><strong>,MF=(E,list addr,COMPLETE)</strong></td>
<td>Default: COMPLETE</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the SAMPLE macro with the following exception:

- **,MF=(E,list addr)**
- **,MF=(E,list addr,COMPLETE)**

  Specifies the execute form of the IOSADMF macro.

  *list addr* specifies the area that the system uses to store the parameters.
  
  COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that you did not specify.
IOSADMF macro
Chapter 86. IOSCAPF — Obtain the actual address of a captured UCB

Description

Use the IOSCAPF macro to obtain the actual address of a specified captured unit control block (UCB) address. A captured UCB is a below 16 megabyte view of an above 16 megabyte UCB. The IOSCAPU macro performs the same function and provides input parameter validation, recovery, and environmental checking. IOSCAPF provides an alternative for passing parameters (that is, in register 1 rather than in a parameter list). IOSCAPU enables you to specify the address of a UCB in another address space. With IOSCAPF, the specified UCB must reside in the current address space.

For information about accessing an above 16 megabyte UCB, see Z/OS MVS Programming: Authorized Assembler Services Guide.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and any PSW key for READ type calls; Supervisor state and any PSW key for CREATE, UPDATE, and DELETE calls.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller may hold locks, but is not required to hold any.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>None.</td>
</tr>
</tbody>
</table>

Programming requirements

The caller must pin the UCB or otherwise guarantee that the UCB will not be deleted.

The caller must supply recovery to handle any unexpected errors, such as abends.

Restrictions

Only use IOSCAPF to translate a captured UCB address that was captured in your primary address space.

Input register information

Before issuing the IOSCAPF macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of UCB common segment of the captured UCB</td>
</tr>
</tbody>
</table>
Before issuing the IOSCAPF macro, the caller does not have to place any information into any access register (AR).

**Output register information**
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>Address of the UCB common segment of the actual UCB</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**
None.

**Syntax**
The standard form of the IOSCAPF macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSCAPF.</td>
</tr>
<tr>
<td>IOSCAPF</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSCAPF.</td>
</tr>
<tr>
<td>MF=(S)</td>
<td>Default: S</td>
</tr>
</tbody>
</table>

**Parameters**
The parameters are explained as follows:

**MF=(S)**
Specifies the standard form of the macro. This parameter is optional.
ABEND codes
None.

Return and reason codes
None.
Chapter 87. IOSCAPU — Capture, release, or obtain the actual address of a UCB

Description

Use the IOSCAPU macro to access an above 16 megabyte unit control block (UCB) with a 24-bit address. IOSCAPU creates a view into the actual above 16 megabyte UCB in below 16 megabyte private storage, which is known as capturing the UCB. An above 16 megabyte UCB is automatically captured at allocation and released at deallocation. You can also use IOSCAPU to explicitly capture and release an above 16 megabyte UCB if necessary.

IOSCAPU enables you to perform the following functions:

- Capture an actual UCB into the private storage area of an address space and receive the captured UCB address with the CAPTUCB option. You can also capture the UCB into common storage.
- Release a captured UCB at a specific address with the UCAPTUCB option
- Receive the 31-bit above 16 megabyte actual address for a specified captured address with the CAPTOACT option.

The environment, programming requirements, restrictions, input register information, output register information, and performance implications generally apply to all the functions. Any exceptions are noted. The syntax, return and reason codes, abend codes, examples, and forms are described in a separate section for each function. See "Capture an UCB function" on page 927, "Release a captured UCB function" on page 932, and "Translate captured to actual address function" on page 937.

Similar to IOSCAPU with the CAPTOACT option, the IOSCAPF macro obtains the above 16 megabyte address of a captured UCB. IOSCAPF enables you to pass the captured UCB address in register 1 rather than in a parameter list but does not provide input parameter validation or recovery.

For information about accessing an above 16 megabyte UCB, see "z/OS MVS Programming: Authorized Assembler Services Guide".

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>CAPTUCB or UCAPTUCB option: problem state with PSW key 0-7, or supervisor state</td>
</tr>
<tr>
<td></td>
<td>CAPTOACT option with ASID: problem state with PSW key 0-7, or supervisor state</td>
</tr>
<tr>
<td></td>
<td>CAPTOACT option without ASID: problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
</tbody>
</table>
IOSCAPU macro

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross memory mode:</td>
<td>CAPTUCB or UCASTUCB option: PASN=HASN=SASN</td>
</tr>
<tr>
<td></td>
<td>CAPTOACT option: any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or Access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>CAPTUCB or UCASTUCB option with CAPTCOM=NO: enabled for I/O and external interrupts</td>
</tr>
<tr>
<td></td>
<td>CAPTUCB option with CAPTCOM=NEVER: enabled for I/O interrupts.</td>
</tr>
<tr>
<td></td>
<td>CAPTUCB or UCASTUCB option with CAPTCOM=YES: enabled or disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td></td>
<td>CAPTOACT option: enabled or disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>CAPTUCB or UCASTUCB option with CAPTCOM=NO: no locks held.</td>
</tr>
<tr>
<td></td>
<td>CAPTUCB option with CAPTCOM=NEVER: no locks held.</td>
</tr>
<tr>
<td></td>
<td>CAPTUCB or UCASTUCB option with CAPTCOM=YES: the caller may hold locks, but is not required to hold any.</td>
</tr>
<tr>
<td></td>
<td>CAPTOACT option: the caller may hold locks, but is not required to hold any.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>If the caller of IOSCAPU with the CAPTOACT option is disabled, the parameter list must be in nonpageable or disabled reference (DREF) storage. This situation is also true for a caller of IOSCAPU with the CAPTUCB or UCASTUCB option and the CAPTCOM=YES parameter.</td>
</tr>
</tbody>
</table>

Programming requirements
The caller must pin the UCB or otherwise guarantee that the UCB will not be dynamically deleted.

Restrictions
Only use IOSCAPU CAPTOACT without ASID, to translate a captured UCB address that was captured in your primary address space.

Input register information
Before issuing the IOSCAPU macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>
Return code

When control returns to the caller, the ARs contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
</tr>
<tr>
<td>Unchanged</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Capture an UCB function

Syntax

The standard form of the IOSCAPU macro with the CAPTUCB option is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td></td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>One or more blanks must precede IOSCAPU.</td>
</tr>
<tr>
<td>IOSCAPU</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>One or more blanks must follow IOSCAPU.</td>
</tr>
<tr>
<td>CAPTUCB</td>
<td>,UCBPTR=ucbptr :ucbptr :RS-type or address in register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td>,CAPTPTR=captptr :captptr :RS-type or address in register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td>,LASTING=NO Default: LASTING=NO</td>
</tr>
<tr>
<td></td>
<td>,LASTING=YES</td>
</tr>
<tr>
<td></td>
<td>,CAPTCOM=NO Default: CAPTCOM=NO</td>
</tr>
<tr>
<td></td>
<td>,CAPTCOM=YES</td>
</tr>
<tr>
<td></td>
<td>,CAPTCOM=NEVER</td>
</tr>
<tr>
<td></td>
<td>,LINKAGE=SYSTEM Default: LINKAGE=SYSTEM</td>
</tr>
</tbody>
</table>
### IOSCAPU macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,LINKAGE=BRANCH</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode addr</td>
<td>retcode addr : RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,RSNCODE=rsncode addr</td>
<td>rsncode addr : RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

- **CAPTUCB**
  Specifies that you want to capture an actual UCB into the private storage area of an address space. Capturing the UCB enables you to access the UCB with a 24-bit address.

  ,UCBPTR=ucbptr
  Specifies a pointer that contains the address of the common segment of the actual UCB that you want to capture.

- **CAPTPTR=captptr**
  Specifies the pointer to contain the address of the common segment of the captured UCB.

  **Note:** CAPTPTR is a four byte field. If the caller specifies a field with a high order byte for flags, those flags are overlaid when the macro specifies the output pointer to the captured UCB.

- **LASTING=YES, LASTING=NO**
  Specifies whether the system should release the captured UCB automatically during end of task termination.
  - **NO:** Frees any captured UCBs during the end of job step task
  - **YES:** Leaves any captured UCBs during the end of job step task

  **Note:** If, and only if, LASTING=YES is specified when capturing a UCB, LASTING=YES should be specified when releasing the same captured UCB.

- **CAPTCOM=NO, CAPTCOM=YES, CAPTCOM=NEVER**
  Specifies whether the above 16 megabyte UCB should be captured into common storage.
  - **NO:** Capture the UCB into private storage of the current address space
  - **YES:** Capture the UCB into common storage. This option is not recommended because it uses common storage.
  - **NEVER:** Unconditionally capture the UCB into private storage of the current address space.

  **Note:**
IOSCAPU macro

Since there are reasons why a CAPTCOM=NO request may still cause a UCB to be captured to common (i.e., if the UCB is already captured in common), this keyword can be used to force IOS to capture the UCB to private storage.

Specifying CAPTCOM=NEVER may cause duplicate UCB storage to be allocated in the case where the UCB is already captured to common.

Since captures done in MASTERS address space will always be captured to common storage, the CAPTCOM=NEVER specification will be ignored in this case.

\[ LINKAGE=SYSTEM \]
\[ LINKAGE=BRANCH \]

Specifies the type of call that should be generated:
- \[ SYSTEM \]: Specifies a program call (PC)
- \[ BRANCH \]: Specifies a branch entry

\[ RETCODE=\text{retcode addr} \]

Specifies the location where the system is to store the return code. The return code is also in GPR 15.

\[ RSNCODE=\text{rsncode addr} \]

Specifies the location where the system is to store the reason code. The reason code is also in GPR 0.

ABEND codes

IOSCAPU might abnormally end with abend code X'2C6'. See \[ EOS MVS System Codes \] for an explanation of abend code X'2C6'.

Return and reason codes

When the IOSCAPU macro returns control to your program, GPR 15 (and retcode if you coded RETCODE) contains the return code. If the return code is not 0, GPR 0 (and rsncode if you coded RSNCODE) contains the reason code.

\[ Table 80. Return and Reason Codes for the IOSCAPU CAPTUCB Macro \]

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None</td>
<td>Meaning: IOSCAPU completed successfully. Action: None.</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td>Meaning: Warning. The program attempted to capture a below 16 megabyte UCB. The address of the actual UCB is returned and a capture is not performed. Action: None required if the program attempts to capture any input UCB. Otherwise, check the address of the actual UCB. Correct the error and rerun the program.</td>
</tr>
<tr>
<td>08</td>
<td>0C</td>
<td>Meaning: Warning. The program attempted to capture a UCB that was at a captured UCB address. Action: None required if the program attempts to capture any input UCB. Otherwise, check to see if your program correctly specified the actual UCB address on the UCBPTR parameter. Correct the error and rerun the program.</td>
</tr>
</tbody>
</table>
Table 80. Return and Reason Codes for the IOSCAPU CAPTUCB Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08                      | 10                      | **Meaning**: Program error. The program attempted to use a UCB address that is not a valid UCB.  
**Action**: Check to see if your program correctly specified the UCB address on the UCBPTR parameter. Correct the error and rerun the program. |
| 20                      |                         | **Meaning**: System error. This return code is for IBM diagnostic purposes only. Most likely, the system could not obtain storage that it required.  
**Action**: Record the return code and supply it to the appropriate IBM support personnel. |

**Example**

Capture a UCB at the address specified by UCBPTR and receive the captured UCB address in CAPTURED.

```assembly
IOS_CAPT IOSCAPU CAPTUCB, X
UCBPTR=UCBPTR, X
CAPTPTR=CAPTURED, X
LINKAGE=BRANCH, X
MF=(E,CAPTLIST)
```

**IOSCAPU CAPTUCB - List form**

Use the list form of the IOSCAPU macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

**Syntax**

The list form of the IOSCAPU macro with the CAPTUCB option is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSCAPU.</td>
</tr>
<tr>
<td>IOSCAPU</td>
<td></td>
</tr>
<tr>
<td><code>list addr</code></td>
<td><code>list addr</code>: symbol.</td>
</tr>
<tr>
<td><code>attr</code></td>
<td><code>attr</code>: 1- to 60-character input string.</td>
</tr>
<tr>
<td><code>0D</code></td>
<td>Default: <code>0D</code></td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained under the standard form of the IOSCAPU macro with the following exception:

- \texttt{MF=(L,list addr)}
- \texttt{MF=(L,list addr,attr)}
- \texttt{MF=(L,list addr,0D)}

Specifies the list form of the IOSCAPU macro.

\textit{list addr} is the name of a storage area to contain the parameters.

\textit{attr} is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code \textit{attr}, the system provides a value of X'0D', which forces the parameter list to a doubleword boundary.

**IOSCAPU CAPTUCB - Execute form**

Use the execute form of the IOSCAPU macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

**Syntax**

The execute form of the IOSCAPU macro with the CAPTUCB option is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{name}</td>
<td>\textit{name}: symbol. Begin \textit{name} in column 1.</td>
</tr>
<tr>
<td>\texttt{b}</td>
<td>One or more blanks must precede IOSCAPU.</td>
</tr>
<tr>
<td>IOSCAPU</td>
<td></td>
</tr>
<tr>
<td>\texttt{b}</td>
<td>One or more blanks must follow IOSCAPU.</td>
</tr>
<tr>
<td>CAPTUCB</td>
<td></td>
</tr>
<tr>
<td>\texttt{,UCBPR=pcbptr}</td>
<td>\textit{ucbptr}:RS-type or address in register (2) - (12).</td>
</tr>
<tr>
<td>\texttt{,CAPTPTR=cpptr}</td>
<td>\textit{cpptr}:RS-type or address in register (2) - (12).</td>
</tr>
<tr>
<td>\texttt{,LASTING=NO}</td>
<td>Default: LASTING=NO</td>
</tr>
<tr>
<td>\texttt{,LASTING=YES}</td>
<td></td>
</tr>
</tbody>
</table>
### IOSCAPU macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,CAPTCOM=NO</td>
<td>Default: CAPTCOM=NO</td>
</tr>
<tr>
<td>,CAPTCOM=YES</td>
<td></td>
</tr>
<tr>
<td>,LINKAGE=SYSTEM</td>
<td>Default: LINKAGE=SYSTEM</td>
</tr>
<tr>
<td>,LINKAGE=BRANCH</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode addr</td>
<td>retcode addr: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,RSNCODE=rsncode addr</td>
<td>rsncode addr: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td>list addr: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td>Default: COMPLETE</td>
</tr>
<tr>
<td>,MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the IOSCAPU macro with the following exception:

- ,MF=(E,list addr)
- ,MF=(E,list addr,COMPLETE)
- ,MF=(E,list addr,NOCHECK)

Specifies the execute form of the IOSCAPU macro.

- list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

### Release a captured UCB function

#### Syntax

The standard form of the IOSCAPU macro with the UCAPTUCB option is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>Δ</td>
<td>One or more blanks must precede IOSCAPU.</td>
</tr>
<tr>
<td>IOSCAPU</td>
<td></td>
</tr>
</tbody>
</table>
### Parameters

The parameters are explained as follows:

**UCAPTUCB**

- Specifies that you want a captured UCB released.

**\_CAPTPTR=captptr**

- Specifies the address of the common segment of the captured UCB that you want released.

**\_CAPTCOM=NO**

- Specifies whether the above 16 megabyte UCB should be released from common storage.
  - **NO**: Release the UCB from private storage of the current address space
  - **YES**: Release the UCB from common storage.

**\_LASTING=NO**

- This is **NOT** a UCAPTUCB associated with a previous CAPTUCB (where LASTING=YES was specified).

**\_LASTING=YES**

- This is a UCAPTUCB associated with a previous CAPTUCB where LASTING=YES was specified.

---

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>△</td>
<td>One or more blanks must follow IOSCAPU.</td>
</tr>
<tr>
<td>UCAPTUCB</td>
<td></td>
</tr>
<tr>
<td>_CAPTPTR=captptr</td>
<td><code>captptr</code>: RS-type or address in register (2) - (12).</td>
</tr>
<tr>
<td>_LASTING=NO</td>
<td><strong>Default</strong>: LASTING=NO</td>
</tr>
<tr>
<td>_LASTING=YES</td>
<td></td>
</tr>
<tr>
<td>_CAPTCOM=NO</td>
<td><strong>Default</strong>: CAPTCOM=NO</td>
</tr>
<tr>
<td>_CAPTCOM=YES</td>
<td></td>
</tr>
<tr>
<td>_LINKAGE=SYSTEM</td>
<td><strong>Default</strong>: LINKAGE=SYSTEM</td>
</tr>
<tr>
<td>_LINKAGE=BRANCH</td>
<td></td>
</tr>
<tr>
<td>_RETCODE=retcode addr</td>
<td><code>retcode addr</code>: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>_RSNCODE=rsncode addr</td>
<td><code>rsncode addr</code>: RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
</tbody>
</table>
IOSCAPU macro

Note: If, and only if, LASTING=YES is specified when capturing a UCB, LASTING=YES should be specified when releasing the same captured UCB.

\texttt{,LINKAGE=SYSTEM}
\texttt{,LINKAGE=BRANCH}
Specifies the type of call that should be generated:
\begin{itemize}
  \item \texttt{SYSTEM}: Specifies a program call (PC)
  \item \texttt{BRANCH}: Specifies a branch entry
\end{itemize}

\texttt{,RETCODE=retcode addr}
Specifies the location where the system is to store the return code. The return code is also in GPR 15.

\texttt{,RSNCODE=rsncode addr}
Specifies the location where the system is to store the reason code. The reason code is also in GPR 0.

ABEND codes
IOSCAPU might abnormally end with abend code X'2C6'. See \texttt{z/OS MVS System Codes} for an explanation of abend code X'2C6'.

Return and reason codes
When the IOSCAPU macro returns control to your program, GPR 15 (and \texttt{retcode} if you coded \texttt{RETCODE}) contains the return code. If the return code is not 0, GPR0 (and \texttt{rsncode} if you coded \texttt{RSNCODE}) contains the reason code.

\textit{Table 81. Return and Reason Codes for the IOSCAPU UCAPTUCB Macro}

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None</td>
<td>\textbf{Meaning}: IOSCAPU completed successfully. \textbf{Action}: None.</td>
</tr>
<tr>
<td>04 08</td>
<td></td>
<td>\textbf{Meaning}: Warning. The program attempted to release an actual below 16 megabyte UCB. \textbf{Action}: None required if the program tries to release any input UCB. Otherwise, check the address of the captured UCB. Correct the error and rerun the program.</td>
</tr>
<tr>
<td>08 08</td>
<td></td>
<td>\textbf{Meaning}: Program error. The program attempted to release a captured UCB and the captured UCB does not exist in the address space. \textbf{Action}: Check to see if your program correctly specified the captured UCB address on the CAPTPTR parameter. Correct the error and rerun the program.</td>
</tr>
<tr>
<td>08 10</td>
<td></td>
<td>\textbf{Meaning}: Program error. The program attempted to use a UCB address that is not a valid UCB. \textbf{Action}: Check to see if your program correctly specified the UCB address on the UCBPTR or CAPTPTR parameter. Correct the error and rerun the program.</td>
</tr>
<tr>
<td>08 18</td>
<td></td>
<td>\textbf{Meaning}: Warning. The program attempted to release an actual above 16 megabyte UCB. \textbf{Action}: None required if the program tries to release any input UCB. Otherwise, check the address of the captured UCB. Correct the error and rerun the program.</td>
</tr>
</tbody>
</table>
Table 81. Return and Reason Codes for the IOSCAPU UCAPTUCB Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td></td>
<td><strong>Meaning</strong>: System error. This return code is for IBM diagnostic purposes only. Most likely, the system could not obtain storage that it required. <strong>Action</strong>: Record the return code and supply it to the appropriate IBM support personnel.</td>
</tr>
</tbody>
</table>

**Example**

Release the captured UCB at the address specified by CAPTURED.

```plaintext
IOS_UNCA IOSCAPU UCAPTUCB, X
CAPTPTR=CAPTURED, X
LINKAGE=BRANCH, X
MF=(E,CAPTLIST)
```

**IOSCAPU UCAPTUCB - List form**

Use the list form of the IOSCAPU macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

**Syntax**

The list form of the IOSCAPU macro with the UCAPTUCB option is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSCAPU.</td>
</tr>
<tr>
<td>IOSCAPU</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSCAPU.</td>
</tr>
<tr>
<td>MF=(L,list addr)</td>
<td><code>list addr</code>: symbol.</td>
</tr>
<tr>
<td>MF=(L,list addr,attr)</td>
<td><code>attr</code>: 1- to 60-character input string.</td>
</tr>
<tr>
<td>MF=(L,list addr,0D)</td>
<td><strong>Default</strong>: 0D</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained under the standard form of the IOSCAPU macro with the following exception:

`MF=(L,list addr)`
IOSCAPU macro

\[ MF=(L, list\ addr, attr) \]
\[ MF=(L, list\ addr, 0D) \]

Specifies the list form of the IOSCAPU macro.

\[ list\ addr \] is the name of a storage area to contain the parameters.

\[ attr \] is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code \[ attr \], the system provides a value of \[ '0D' \], which forces the parameter list to a doubleword boundary.

IOSCAPU UCAPTUCB - Execute form

Use the execute form of the IOSCAPU macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the IOSCAPU macro with the UCAPTUCB option is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSCAPU.</td>
</tr>
<tr>
<td>IOSCAPU</td>
<td>One or more blanks must follow IOSCAPU.</td>
</tr>
<tr>
<td>UCAPTUCB</td>
<td></td>
</tr>
<tr>
<td>,CAPTPTR=capptr</td>
<td>capptr : RS-type or address in register (2) - (12).</td>
</tr>
<tr>
<td>,CAPTCOM=NO</td>
<td>Default: CAPTCOM=NO</td>
</tr>
<tr>
<td>,CAPTCOM=YES</td>
<td></td>
</tr>
<tr>
<td>,LINKAGE=SYSTEM</td>
<td>Default: LINKAGE=SYSTEM</td>
</tr>
<tr>
<td>,LINKAGE=BRANCH</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode addr</td>
<td>retcode addr : RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,RSNCODE=rsncode addr</td>
<td>rsncode addr : RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,MF=(E, list addr)</td>
<td>list addr: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,MF=(E, list addr, COMPLETE)</td>
<td>Default: COMPLETE</td>
</tr>
</tbody>
</table>
### Parameters

The parameters are explained under the standard form of the IOSCAPU macro with the following exception:

- \( \text{MF} = (E, \text{list addr}) \)
- \( \text{MF} = (E, \text{list addr}, \text{COMPLETE}) \)
- \( \text{MF} = (E, \text{list addr}, \text{NOCHECK}) \)

Specifies the execute form of the IOSCAPU macro.

- \( \text{list addr} \) specifies the area that the system uses to contain the parameters.

- COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

- NOCHECK specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

### Translate captured to actual address function

#### Syntax

The standard form of the IOSCAPU macro with the CAPTOACT option is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{name} )</td>
<td>( \text{name} ): symbol. Begin ( \text{name} ) in column 1.</td>
</tr>
<tr>
<td>( \text{b} )</td>
<td>One or more blanks must precede IOSCAPU.</td>
</tr>
<tr>
<td>IOSCAPU</td>
<td></td>
</tr>
<tr>
<td>( \text{b} )</td>
<td>One or more blanks must follow IOSCAPU.</td>
</tr>
<tr>
<td>CAPTOACT</td>
<td></td>
</tr>
<tr>
<td>( \text{CAPPTR} = \text{captptr} )</td>
<td>( \text{captptr} ): RS-type or address in register (2) - (12).</td>
</tr>
<tr>
<td>( \text{UCPTR} = \text{ucbptr} )</td>
<td>( \text{ucbptr} ): RS-type or address in register (2) - (12).</td>
</tr>
<tr>
<td>( \text{ASID} = \text{CURRENT} )</td>
<td>\text{Default: ASID} = \text{CURRENT}</td>
</tr>
<tr>
<td>( \text{ASID} = \text{asid} )</td>
<td>( \text{asid} ): RS-type or address in register (2) - (12).</td>
</tr>
<tr>
<td>( \text{LINKAGE} = \text{SYSTEM} )</td>
<td>\text{Default: LINKAGE} = \text{SYSTEM}</td>
</tr>
<tr>
<td>( \text{LINKAGE} = \text{BRANCH} )</td>
<td></td>
</tr>
</tbody>
</table>
IOSCAPU macro

### Parameters

The parameters are explained as follows:

**CAPTOACT**
- Specifies that you want the actual UCB address for a captured UCB.

**,CAPTPTR=captptr**
- Specifies the pointer to the address of the common segment of the captured UCB.

**,UCBPTR=ucbptr**
- Specifies a pointer to contain the address of the actual UCB common segment.

**,ASID=CURRENT**, ,ASID=asid
- Specifies the address space in which the captured UCB was originally captured.
  - CURRENT: Specifies the address space of the program
  - asid: Specifies the name of another address space

**,LINKAGE=SYSTEM**, ,LINKAGE=BRANCH
- Specifies the type of call that should be generated:
  - SYSTEM: Specifies a program call (PC)
  - BRANCH: Specifies a branch entry

**,RETCODE=retcode addr**
- Specifies the location where the system is to store the return code. The return code is also in GPR 15.

**,RSNCODE=rsncode addr**
- Specifies the location where the system is to store the reason code. The reason code is also in GPR 0.

### ABEND codes

IOSCAPU might abnormally end with abend code X'2C6'. See [z/OS MVS System Codes](#) for an explanation of abend code X'2C6'.

### Return and reason codes

When the IOSCAPU macro returns control to your program, GPR 15 (and retcode if you coded RETCODE) contains the return code. If the return code is not 0, GPR0 (and rsncode if you coded RSNCODE) contains the reason code.

---

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,RETCODE=retcode addr</td>
<td>retcode addr : RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
<tr>
<td>,RSNCODE=rsncode addr</td>
<td>rsncode addr : RS-type address or address in register (2) - (12) of fullword output variable.</td>
</tr>
</tbody>
</table>
Table 82. Return and Reason Codes for the IOSCAPU CAPTOACT Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | None                    | **Meaning:** IOSCAPU completed successfully.  
                          |                          | **Action:** None. |
| 04                      | 0C                      | **Meaning:** Warning. The program requested the actual address of an actual UCB.  
                          |                          | **Action:** None required if the program always attempts to receive the actual UCB address for a UCB. Otherwise, check the address of the captured UCB. Correct the error and rerun the program. |
| 08                      | 04                      | **Meaning:** Program error. The program attempted to receive the actual UCB address for a captured UCB and the address space identifier specified for the captured UCB does not exist or the address space was swapped out.  
                          |                          | **Action:** Retry the request because the address space might have been swapped in. Also, check to see if your program correctly specified the address space of the captured UCB on the ASID parameter. Correct the error and rerun the program. |
| 08                      | 10                      | **Meaning:** Program error. The program attempted to use a UCB address that is not a valid UCB.  
                          |                          | **Action:** Check to see if your program correctly specified the UCB address on the CAPTPTR parameter. Correct the error and rerun the program. |
| 20                      |                         | **Meaning:** System error. This return code is for IBM diagnostic purposes only. Most likely, the system could not obtain storage that it required.  
                          |                          | **Action:** Record the return code and supply it to the appropriate IBM support personnel. |

**Example**

Receive the actual UCB address (in ACTUAL) of the captured UCB address specified by CAPTURED.

IOS_TRAN IOSCAPU CAPTOACT, X
UCBPR=ACTUAL, X
CAPTPR=CAPTURED, X
LINKAGE=BRANCH, X
MF=(E,CAPTLIST)

**IOSCAPU CAPTOACT - List form**

Use the list form of the IOSCAPU macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

**Syntax**

The list form of the IOSCAPU macro with the CAPTOACT option is written as follows:
**IOSCAPU macro**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede IOSCAPU.</td>
</tr>
<tr>
<td>IOSCAPU</td>
<td></td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must follow IOSCAPU.</td>
</tr>
<tr>
<td><code>MF=(L,list addr)</code></td>
<td><code>list addr</code>: symbol.</td>
</tr>
<tr>
<td><code>MF=(L,list addr,attr)</code></td>
<td><code>attr</code>: 1- to 60-character input string.</td>
</tr>
<tr>
<td><code>MF=(L,list addr,0D)</code></td>
<td>Default: <code>0D</code>.</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained under the standard form of the IOSCAPU macro with the following exception:

- `MF=(L,list addr)`
- `MF=(L,list addr,attr)`
- `MF=(L,list addr,0D)`
  
  Specifies the list form of the IOSCAPU macro.
  
  `list addr` is the name of a storage area to contain the parameters.
  
  `attr` is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code `attr`, the system provides a value of `X'0D'`, which forces the parameter list to a doubleword boundary.

**IOSCAPU CAPTOACT - Execute form**

Use the execute form of the IOSCAPU macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

**Syntax**

The execute form of the IOSCAPU macro with the CAPTOACT option is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede IOSCAPU.</td>
</tr>
<tr>
<td>IOSCAPU</td>
<td></td>
</tr>
</tbody>
</table>
Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSCAPU.</td>
</tr>
</tbody>
</table>

CAPTOACT

,UCBPTR=ucbptr  
ucbptr : RS-type or address in register (2) - (12).

,CAPTPTR=capptr  
capptr : RS-type or address in register (2) - (12).

,ASID=CURRENT  
Default: ASID=CURRENT

,ASID=asid  
asid : RS-type or address in register (2) - (12).

,LINKAGE=SYSTEM  
Default: LINKAGE=SYSTEM

,LINKAGE=BRANCH

,RETCODE=retcode addr  
retcode addr : RS-type address or address in register (2) - (12) of fullword output variable.

,RSNCODE=rsncode addr  
rsncode addr : RS-type address or address in register (2) - (12) of fullword output variable.

,MF=(E,list addr)  
list addr: RX-type address or address in register (2) - (12).

,MF=(E,list addr,COMPLETE)  
Default: COMPLETE

,MF=(E,list addr,NOCHECK)

Parameters

The parameters are explained under the standard form of the IOSCAPU macro with the following exception:

,MF=(E,list addr)  

,MF=(E,list addr,COMPLETE)

,MF=(E,list addr,NOCHECK)

Specifies the execute form of the IOSCAPU macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.
IOSCAPU macro
Chapter 88. IOSCDR — Retrieve configuration data records

Description

The IOSCDR macro enables authorized callers to retrieve device identification information (such as the serial number and the model number) for an I/O device located along a specific I/O path. This information can allow installation management to do the following:

- Uniquely identify, across multiple systems, I/O hardware located along a specific I/O path
- Following device installs, check device paths to ensure that cables are connected to the proper device before bringing the device or path online
- Construct a map of an installation's configuration
- During problem diagnosis, ensure that all paths to a given device are reaching the expected device.

The configuration data record (CDR) information that IOSCDR retrieves is mapped by the mapping macro IHACDR.

The format of IHACDR is in z/OS MVS Data Areas, Vol 3 (IEFDORC-ISGYQCBP).

For more information about the contents of CDRs and information about the contents of node descriptors (NDs), see ESA/390 Common I/O Device Commands.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Authorization</td>
<td>For LINKAGE=LINK, supervisor state and any PSW key. For LINKAGE=SYSTEM, any one or more of the following:</td>
</tr>
<tr>
<td></td>
<td>• Supervisor state</td>
</tr>
<tr>
<td></td>
<td>• PKM allowing key 0 – 7</td>
</tr>
<tr>
<td></td>
<td>• PSW key 0 – 7</td>
</tr>
<tr>
<td></td>
<td>• APF-authorized</td>
</tr>
<tr>
<td></td>
<td>• RACF authorization to the FACILITY class and the IOSCDR entity</td>
</tr>
</tbody>
</table>

Dispatchable unit mode: Task
Cross memory mode: PASN=HASN=SASN
AMODE: 31-bit
ASC mode: Primary
Interrupt status: Enabled for I/O and external interrupts
Locks: No locks held
Control parameters: Control parameters must be in the primary address space.

Programming requirements

Include the IHACDR mapping macro.

Restrictions

The caller can have no enabled, unlocked task (EUT) FRRs established.
**IOSCDR macro**

Note that, when you issue IOSCDR, the service pins the device so that the device's UCB and other related data structures are not dynamically deleted while IOSCDR is retrieving the data. When IOSCDR completes, it unpins the device.

**Input register information**

Before issuing the IOSCDR macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

**Output register information**

When control returns to the caller of the IOSCDR macro, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by IOSCDR</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by IOSCDR</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller of the IOSCDR macro, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**Syntax**

The standard form of the IOSCDR macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSCDR.</td>
</tr>
<tr>
<td>IOSCDR</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSCDR.</td>
</tr>
</tbody>
</table>
### Parameter descriptions

The parameters are explained as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVN=device num</td>
<td><em>device num</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,SCHSET=xschset</td>
<td><em>xschset</em>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,CHPID=path id</td>
<td><em>path id</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,CDRERA=cdr area</td>
<td><em>cdr area</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,CDRLEN=cdr length</td>
<td><em>cdr length</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,CDRSIZE=cdr size</td>
<td><em>cdr size</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,LINKAGE=SYSTEM Default</td>
<td>LINK</td>
</tr>
<tr>
<td>,LINKAGE=LINK</td>
<td></td>
</tr>
<tr>
<td>,NODE_DESCRIPTOR=node descriptor</td>
<td>Optional input. It is the name (RS-type), or address in register (2)-(12), of the 32 bytes of storage for one node descriptor to be returned by the service. The <em>node descriptor</em> is associated with the control unit that is attached to the specified path in the input mask.</td>
</tr>
<tr>
<td>,READ=NOIO</td>
<td></td>
</tr>
<tr>
<td>,READ=IO</td>
<td></td>
</tr>
<tr>
<td>,READ=COND</td>
<td></td>
</tr>
<tr>
<td>,STATUS=status</td>
<td><em>status</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,TIME=time</td>
<td><em>time</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,IOCTOKEN=ioctoken addr</td>
<td><em>ioctoken addr</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,WWPN=xwwpn</td>
<td><em>xwwpn</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,RETCODE=return code</td>
<td><em>return code</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=reason code</td>
<td><em>reason code</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=parameter list version</td>
<td></td>
</tr>
</tbody>
</table>
DEVN=device num
  Specifies the binary device number (0000 - FFFF) of a device for which
  IOSCDR retrieves a CDR.

SCHSET=xschset
  Specifies the name (RS-type), or address in register (2)-(12), of an optional byte
  input that specifies a subchannel set for the CDR that is to be retrieved.
  DEFAULT: 0.

CHPID=path id
  Specifies the channel path ID (00 - FF) of a specific path for which IOSCDR
  retrieves a CDR. To determine the ID for a specific channel path, use the
  UCBINFO PATHINFO macro or the DISPLAY MATRIX operator command.

CDRAREA=cdr area
  Specifies the name of the work area that receives a copy of the CDR for the
  specified device and path. You must specify on the CDRLEN parameter the
  length of the CDR area. The CDR area is mapped by IHACDR. See z/OS MVS
  Data Areas in the z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkserv/) for more information about IHACDR.

CDRLEN=cdr length
  Specifies the length of the CDR area. The maximum length of the CDR area is
  65535 bytes. You can start with a length of 256 bytes. If the length you specify
  is smaller than the CDR, IOSCDR returns only a partial CDR and the caller
  receives return code X'04' and reason code X'04'. To ensure that IOSCDR
  returned the entire CDR, verify that the value returned on CDRSIZE is less
  than or equal to CDRLEN. Note that CDRSIZE is device dependent.

CDRSIZE=cdr size
  Specifies the area into which IOSCDR returns the actual size of the CDR for
  the specified device and path. You can use this parameter for diagnostic
  purposes to determine how large CDRLEN should be.

LINKAGE=SYSTEM
  LINKAGE=LINK
  Specifies the type of call that should be generated:
  • SYSTEM: Specifies a Program Call (PC) that passes control to the service
    routine. The caller does not have to be in supervisor state.
  • LINK: Specifies a LINK macro call to pass control to the service routine.
    This call is more direct but the caller must be in supervisor state.

NODE_DESCRIPTOR=node descriptor area
  Specifies the name of the work area that receives a copy of the node descriptor
  for the specified device and path.

READ=NOIO
  READ=IO
  READ=COND
  Specifies that IOSCDR retrieves the CDR or ND in one of the following ways:
  • NOIO—IOSCDR retrieves the last CDR or ND known to MVS for the
    specified device and path. Note that this CDR or ND might not exist or
    might not be current if the specified device is offline. If the specified device
    is online, this option is fairly reliable and is quick because no I/O is
    performed.
  • IO—IOSCDR retrieves the CDR or ND for a specified device and path
    directly from the specified device. Although this option is slower than
    READ=NOIO, READ=IO provides more current information.
**IOSCDR macro**

- **COND**— If the specified device and path are online, IOSCDR retrieves the last CDR or ND known to MVS for the specified device and path. Otherwise, IOSCDR retrieves the CDR directly from the device. COND is the best option to choose if you are interested in retrieving the most accurate CDR or ND in the shortest time possible. Note that the READ parameter has no effect on the way in which a WWPN is retrieved; only the last WWPN known to MVS can be returned.

**,STATUS=status**

Specifies a one-byte field containing status information about successful invocations of IOSCDR. The bit positions, represented in hexadecimal values, are as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on</td>
<td>CDR returned was read from the device.</td>
</tr>
<tr>
<td></td>
<td>off</td>
<td>CDR returned was the last CDR known to MVS.</td>
</tr>
<tr>
<td>1</td>
<td>on</td>
<td>Specified CHPID was logically online to the device.</td>
</tr>
<tr>
<td></td>
<td>off</td>
<td>Specified CHPID was logically offline to the device.</td>
</tr>
<tr>
<td>2</td>
<td>on</td>
<td>Specified device was online.</td>
</tr>
<tr>
<td></td>
<td>off</td>
<td>Specified device was offline.</td>
</tr>
<tr>
<td>3 - 7</td>
<td>—</td>
<td>Reserved for IBM use.</td>
</tr>
</tbody>
</table>

**,TIME=time**

Specifies an 8-byte field containing the maximum amount of time, in seconds, that IOSCDR can run before being purged. The default for the TIME parameter is 5 seconds. You can use TIME when you specify READ=COND or READ=IO. When you specify READ=NOIO, IOSCDR ignores the TIME parameter.

The time interval, whose address resides in virtual storage, is presented as zoned decimal digits in the form:

HHMMSSStth, where:

- **HH** is hours (24-hour clock)
- **MM** is minutes
- **SS** is seconds
- **t** is tenths of seconds
- **h** is hundredths of seconds

IOSCDR runs until one of the following occurs:
- IOSCDR completes successfully or unsuccessfully
- The interval that you specify on the TIME=parameter expires
- The MIH interval for the device expires.

Note that the TIME parameter allows you to set an expiration time that is specific to IOSCDR. The MIH interval, however, is used by other services associated with the device. Using the TIME parameter allows you to set an expiration time that is shorter than the MIH interval.

**,IOCTOKEN=ioctoken addr**

Specifies the address of a 48-character area that contains the MVS I/O configuration token that you supply to IOSCDR. You can obtain this token by issuing the IOCINFO macro, which is described in [z/OS MVS Programming: Assembler Services Reference ABE-HSP](#). If the I/O configuration token that is
current when IOSCDR is invoked does not match the token whose address you
supply as input by ioctoken addr, you receive an error return code.

If you set the input IOCTOKEN (specified by ioctoken addr) to binary zeros,
IOSCDR sets IOCTOKEN to the current I/O configuration token.

For information about how you can use the configuration token to detect
configuration changes, see [z/OS MVS Programming: Authorized Assembler
Services Guide](https://www.ibm.com)

**WWPN=**

Specifies the location where the system is to place the Worldwide Port Name
(WWPN) for the port on the control unit for the specified channel path. If the
WWPN is not available, zeroes will be returned.

**RETCODE=**

Specifies the location or register where the system is to place the return code.
The system copies the return code into the location from register 15.

**RNSCODE=**

Specifies the location or register where the system is to place the reason code.
The system copies the reason code into the location from register 0.

**PLISTVER=IMPLIED_VERSION**

**PLISTVER=MAX**

**PLISTVER=**parameter list version

A decimal value in the "1-2" range that specifies the macro version. PLISTVER
determines which parameter list is generated. Note that MAX can be specified
instead of a number, and the parameter list will be the largest size currently
supported. This size might grow from release to release, thus possibly affecting
the amount of storage needed by your program. If your program can tolerate
this, IBM recommends that you always specify MAX when creating the list
form of the parameter list, as that will ensure that the list form parameter list
is always long enough to hold whatever parameters that might be specified on
the execute form.

The default is IMPLIED_VERSION. When PLISTVER is omitted, the default is
the lowest version that allows all of the parameters specified on the invocation
to be processed.

### Return codes

Return and reason codes, in hexadecimal, from the IOSCDR macro are as follows:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None</td>
<td>Meaning: IOSCDR processing completed successfully. IOSCDR successfully retrieved the CDR for the specified device and path.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td>Meaning: IOSCDR cannot retrieve an entire CDR because the CDR area specified was not large enough to receive the CDR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: The size of the CDR area is determined by CDRLEN. If you do not know what length to specify on CDRLEN, use the optional CDRSIZE parameter. If you specified CDRSIZE, IOSCDR returns the size that CDRAREA needs to be. Retry the operation with a CDR area of the same length as the length returned on CDRSIZE for the failing operation.</td>
</tr>
</tbody>
</table>
### Hexadecimal Return Code | Hexadecimal Reason Code | Meaning and Action
---|---|---
04 | 08 | **Meaning:** IOSCDR cannot retrieve the CDR for the specified device and path. If you specified READ=IO, either a subchannel error or an I/O error could be preventing IOSCDR from retrieving the CDR. If you specified READ=NOIO, a subchannel error could be preventing IOSCDR from retrieving the CDR.

**Action:** Further investigation of the problem is required. The D M=DEV command may provide further diagnosis data. For example, a subchannel error may have occurred because the device is not available. Attempt to vary the path online to produce further diagnosis data. See ESCON Error Recovery Concepts and Procedures in an MVS Environment for further problem diagnosis information. If the problem persists, contact your IBM service representative.

04 | 0C | **Meaning:** IOSCDR cannot retrieve the CDR for the specified device and path. I/O was attempted to the device, but the time interval specified on the TIME parameter expired before I/O completed.

**Action:** Verify that the time interval was sufficiently long. Note that the system issues this return code only if the time expired before the device's MIH interval. To determine the MIH interval, use the 'D MIH' command or the MIHQUERY macro.

04 | 10 | **Meaning:** IOSCDR cannot retrieve the CDR for the specified device and path because MVS does not have a last known CDR to return.

**Action:** Use one of the following methods to retrieve a CDR:
- Bring the device and path online. If a CDR is available, the system will store it.
- Retrieve the CDR directly from the device, by issuing the IOSCDR macro with the READ=IO option.

04 | 14 | **Meaning:** IOSCDR cannot retrieve the last known CDR. IOSCDR did not attempt I/O.

**Action:** A system problem exists that prevents any last known CDR from being retrieved. Retry the operation. If the problem persists, contact IBM Software Support.

08 | 04 | **Meaning:** The specified device does not support the channel control words (CCWs) used to obtain configuration data records.

**Action:** None

08 | 08 | **Meaning:** IOSCDR cannot retrieve the CDR because the device number specified on the DEVN parameter is not valid.

**Action:** Verify your program to ensure that the correct device was passed and retry the operation. If the device number is valid, use the IOCTOKEN keyword to ensure that the device is not dynamically changed or deleted.
### Hexadecimal Return Code
<table>
<thead>
<tr>
<th>Hexadecimal Reason Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08                      | 0C                      | **Meaning**: IOSCDR cannot retrieve the CDR because the channel path id on the CHPID parameter is not valid.  
**Action**: Verify your program to ensure that the correct CHPID was passed and retry the operation. Use the IOCTOKEN keyword to ensure that the CHPID for the device was not dynamically changed or deleted. |
| 08                      | 10                      | **Meaning**: IOSCDR cannot retrieve the CDR because the time specified on the TIME keyword is not valid.  
**Action**: Ensure that the time specified contains valid zoned decimal digits that are in the proper range. |
| 08                      | 14                      | **Meaning**: An incorrect CDR length was specified on the CDRLEN keyword.  
**Action**: Verify that CDRLEN is greater than 0 and does not exceed 65535 bytes, then retry the operation. |
| 08                      | 20                      | **Meaning**: IOSCDR cannot retrieve the CDR because the I/O configuration token that is current when IOSCDR is invoked does not match the token whose address is supplied as input by IOCTOKEN. Note that this return code is only valid for callers using the IOCTOKEN keyword.  
**Action**: Ensure that the device number and CHPID are still valid and retry the operation passing a current IOCTOKEN. |
| 08                      | 24                      | **Meaning**: IOSCDR cannot retrieve the CDR because the IOS address space is not yet available.  
**Action**: Retry the operation after the IOS address space is available (master scheduler initialization has completed). |
| 08                      | 28                      | **Meaning**: IOSCDR cannot establish an ESTAE.  
**Action**: Ensure that there is sufficient private area storage, then retry the operation. |
| 08                      | 2Cx                     | **Meaning**: The value specified on the SCHSET keyword is not valid.  
**Action**: Supply the correct value on the SCHSET keyword. |
| 0C                      | None                    | **Meaning**: An unexpected error occurred.  
**Action**: Record the return code and supply it to the appropriate IBM support personnel. |

### Example
Assume you want to retrieve a configuration data record (CDR) to determine if the manufacturer of a SYSRES volume is IBM.

Scan through all UCBs using the UCBSCAN macro, and put copies of the DASD UCBs the program finds in a user-supplied work area called UCBSTOR. When the program finds the SYSRES device, issue the UCBINFO macro to obtain information.
about the device path and type of channel path for the specified UCB. Information, such as the channel path ID and online status, will appear in the IOSDPATH data area. The program looks through the channel path information until it finds an online path, then issues the IOSCDR macro to retrieve the CDR containing information about the manufacturer of the SYSRES volume.

*.................................................................*
* REGISTER ASSIGNMENTS                          *
*.................................................................*
R0   EQU 0
R1   EQU 1
R2   EQU 2
R3   EQU 3
R4   EQU 4
R5   EQU 5
R6   EQU 6   Dynamic area register
UCBPTR7 EQU 7   UCB Pointer
R8   EQU 8
R9   EQU 9   Module base register
R10  EQU 10
R11  EQU 11
R12  EQU 12
R13  EQU 13   Pointer to standard save area
R14  EQU 14
R15  EQU 15

SPACE 3
TITLE 'IOSSCDRE - IOSCDR Sample Program'

*.................................................................*
* Standard Entry Linkage                          *
*.................................................................*
PRINT GEN
USING *,R9 Sets up base register
ENTRY STM R14,R12,12(R13) Save caller's registers
LR R9,R15 Establish module base register
MODESET KEY=ZERO,MODE=SUP
LA R8,DYNSIZE Load length of dynamic area
STORAGE OBTAIN,LENGTH=((R8)),SP=233 Gets dynamic area
LR R6,R1 Gets dynamic area address
USING DYNAREA,R6 Sets up dynamic area
ST R13,SAVE+4 Save caller's save area address
LA R15,SAVE Get this module's save area address
ST R15,8(R13) Save this module's save area address
* in caller's save area.
LR R13,R15 Set up addressability to this
* module's save area.

B MAINLINE
DC CL8'IOSSCDRE'
DC CL8'&SYSDATE'
DC CL8'&SYSTIME'
TITLE 'IOSSCDRE - SCDRE mainline '

*.................................................................*
* MAINLINE                                          *
*.................................................................*
MAINLINE DS 0H
*  
L 10,X'10' Load CVT pointer
USING CVT,10
TM CVTDCB,CVTOSEXT Is the OLEVEL extension present
BNO NO_IOSCDR No, pre-MVS/SP Version 3 system
*  
TM CVTOSLV1,CVTH5510 Running on version HBB5510?
BNO NO_IOSCDR No, pre-HBB5510 system. IOSCDR
Set up addressability to a storage area called UCBSTOR into which the UCBSCAN macro will return the UCBs of devices it locates.

```assembly
LA UCBPTR7,UCBSTOR  Get address of work area
USING UCB,UCBPTR7   Set up addressability
```

Clear the UCBSCAN work area.

```assembly
LA R0,SCANWORK      Set storage address
LA R1,100          Set storage length
SR R15,R15         Clear second operand
MVCL R0,R14        Clear the storage
```

Loop through all DASD UCBs looking for the SYSRES volume.

Note: There must be a SYSRES volume, and hence it will be found in the scan loop which follows.

```assembly
SCANLOOP UCBSCAN COPY, X
          WORKAREA=SCANWORK, X
          UCBAREA=UCBSTOR, X
          DEVCLASS=DASD, X
          MF=(E,SCANLIST)
          X
```

If UCBSCAN returned a UCB, check whether it is the SYSRES volume. If it isn’t, continue checking more UCBs. If the UCB represents the SYSRES device, end the loop.

```assembly
LTR R15,R15        Test return code
BNZ EXIT_ERROR    Exit if non-zero
TM UCBSTAT,UCBSYSR Test if SYSRES volume
BZ SCANLOOP       Keep looping if not
```

Issue the UCBINFO macro to obtain path-related information. UCBINFO returns this information in a field called PATHSTOR, mapped by IOSDPATH.

Note: Since the device whose path information is sought is the SYSRES device, an online path is certain to be found. No loop counter is used.

```assembly
UCBINFO PATHINFO, X
          DEYN=UCBCHAN, X
          PATHAREA=PATHSTOR, X
          MF=(E,INFOLIST)
          X
```

If UCBINFO cannot retrieve path-related information, that is, you receive a non-zero return code, exit program.

```assembly
LTR R15,R15        Test for 0 return code
BNZ EXIT_ERROR    Exit if bad RC
```
Chapter 88. IOSCDR — Retrieve configuration data records

IOSCDR macro

*.....................................................................*
* Loop through the channel path ID array entries returned in *
* PATHSTOR to find the first online path. An online path *
* is represented by a flag in the array. *
*.....................................................................*
* LA R10,PATHSTOR Address of PATHINFO data *
* USING PATH,R10 Set up addressability to *
* path information. *
* SR R8,R8 CHPID array index register. *

CHPID_LOOP: IC R11,PATHBITS(R8) Get flags from array entry. *
STC R11,PATHSAVE Save entry *
TM PATHSAVE,X'04' Test if the path is online *
BO CHPID_EXIT If so, exit the loop *
LA R8,L'PATHCHPIDARRAY(R8) Increment array index *
B CHPID_LOOP

CHPID_EXIT: LH R11,PATHCHPID(R8) Get the ID for the online *
* channel path. *
STC R11,CHPID Save the ID for the online *
* channel path. *
*.....................................................................*
* The program identifies an online channel path to the SYSRES *
* volume. *
* Issue the IOSCDR macro to request a configuration data *
* record (CDR) for the SYSRES volume whose binary number *
* you specify in the UCBCHAN field. IOSCDR returns the CDR *
* in a storage area called CDRSTOR, whose length you specify *
* on the CDRLEN parameter. *
* Specify the channel path ID (CHPID) of the online *
* path returned by the UCBINFO macro. Also specify *
* the IOSCDR READ=NOIO option to avoid performing *
* I/O operations to the SYSRES volume. The IOSCDR READ=NOIO *
* option will have a CDR to return if the device *
* supports the self-description channel control words (CCWs). *
*.....................................................................*
* IOSCDR DEVN=UCBCHAN, *
* CHPID=CHPID, *
* READ=NOIO, *
* CDRAREA=CDRSTOR, *
* CDRLEN=CDRLEN, *
* CDRSIZE=CDRSIZE, *
* MF=(E,CDRLIST) *
*.....................................................................*
* Check for a zero return code, indicating that IOSCDR completed *
* successfully. If it was not successful, examine the return *
* and reason codes to determine the cause. *
* Note: A large CDRAREA was specified for the purposes of this *
* example to reduce the possibility of the CDRAREA being *
* too small to contain the returned CDR. It is *
* expected that in practical applications of the IOSCDR *
* service, users will obtain the CDRAREA by issuing the *
* GETMAIN macro. If the IOSCDR macro indicates *
* through return and reason codes that the *
* area passed was too small, issue the FREEMAIN macro to *
* release the storage, and obtain a larger area. Reissue *
* the IOSCDR macro. IOSCDR indicates the minimum size *
* for the CDRAREA through the CDRSIZE keyword. *
*.....................................................................*
* LTR R15,R15 Test for 0 return code *
BNZ EXIT_ERROR Exit if bad RC

*.....................................................................*
**IOSCDR macro**

* Scan the CDR, mapped by IHACDR, searching for the node element (NED) for the SYSRES volume. The NEDTCU field should indicate that this device is a control unit.

```
LA R10,CDRSTOR Set up addressability to the
CDRAREA.
USING NED,R10
SR RB,R8 Clear NED index register.
CDR_LOOP TM NEDFLAGS,CDRFNED Check if the record represents an NED.
BNE CDR_ITERATE If not, try next record.
CLI NEDTYPE,NEDTCU Check if the NED represents a control unit.
BNE CDR_ITERATE If not, try next record.
B CDR_EXIT CU NED found.
CDR_ITERATE LA R8,32(R8) Increment index register.
LA R10,32(R10) Increment to next record in CDR.
CL R8,CDRSIZE Make sure that there are more records.
BL CDR_LOOP Iterate loop.
B EXIT_ERROR No CU NED found. Exit program.
```

* If the program finds the NED, check if IBM manufactured the control unit by looking in the NEDMANUF field of the returned CDR. Check if the control unit was manufactured by IBM. Return a WTO to the user describing the result.

```
CDR_EXIT DS 0D
CLC NEDMANUF,=CL3'IBM' Check if built by IBM
BNE NOT_IBM
B IS_IBM
IS_IBM DS 0D
WTO 'IOSSCDRE-CONTROL UNIT FOR SYSRES WAS BUILT BY IBM', X
ROUTCDE=(11),DESC=(2)
B EXIT

NOT_IBM DS 0D
WTO 'IOSSCDRE-CONTROL UNIT FOR SYSRES WAS NOT BUILT BY IBM',X
ROUTCDE=(11),DESC=(2)

BL EXIT
```

* Return a WTO to the user saying that the IOSCDR macro is not available on the system executing this sample program.

```
NO_IOSCDR DS 0H
WTO 'IOSSCDRE - IOSCDR SUPPORTED IN HBB5510 AND HIGHER', X
ROUTCDE=(11),DESC=(2)
B EXIT
```

* Return a WTO to the user saying that the IOSCDR macro encountered an error during execution of this sample program.

```
EXIT_ERROR DS 0H
WTO 'IOSSCDRE - THE SAMPLE ENCOUNTERED AN ERROR', X
ROUTCDE=(11),DESC=(2)
```

* Clean up and exit.
IOSCDR macro

```
*.....................................................................*
EXIT       DS  0H
             L    R13,SAVE+4Reloads caller's save
                      area addr into 11
             LA    R0,DYNSIZELoads dynamic area size
             STORAGE  RELEASE,SP=233,ADDR=(R6),LENGTH=(R0)
             MODESET  KEY=NZERO,MODE=PROB
             LM    R14,R12,12(R13)Loads return regs
             BR    R14Returns to caller

* Define constants
* *
CDRLEN    DC F'512'
*
* DSECTs to map save areas and dynamic area
*
* *
DYNSTART  DS  0H
DYNArea    DSECT
*       Save area
SAVE      DS  18F
             DS  0D                      Force doubleword alignment
             SPACE  2
*
*       Issue the list forms of macros since the module is reentrant.
*
* *
LIST_INFOSERV  UCBIINFO MF=(L,INFOLIST) List form of UCBIINFO
INFOSERV_END DS  0D
PATHSTOR    DS  CL256Storage for the PATHAREA
PATHSTOR_END DS  0D
LIST_CDRSERV IOSCDR MF=(L,CDRLIST) List form of IOSCDR
CDRSERV_END DS  0D
CDRSTOR     DS  CL512Storage for the CDRAREA
CDRSTOR_END DS  0D
LIST_SCANSERV UCBSCAN MF=(L,SCANTLIST) List form of UCBSCAN
SCANSERV_END DS  0D
SCANWORK    DS  CL100Scan work area
SCANWORK_END DS  0D
UCBSTOR     DS  CL48UCB copy storage
UCBSTOR_END DS  0D
*
*       Work variables and data structures local to this module
*
* *
CDRSIZE    DS  F                         Actual size of CDR
CHPID      DS  C                         CHPID used for IOSCDR invocation
PATHSAVE   DS  C                         Work variable for CHPID array
*       entries in the PATHAREA.
END_DYN    DS  0D
DYNSIZE    EQU *=DYNAreaCalculates Dynamic area
*
* DSECTs
*
* *
IOSSCDRE   CSECT                        TITLE 'IOSSCDRE - DSECT MAPPINGS'
EJECT
```

Chapter 88. IOSCDR — Retrieve configuration data records 955
IOSCDR macro

    CVT LIST=YES,DSECT=YES
* 
UCB DSECT
   IEFUCBOB
*
CDRAREA IHACDR DSECT=YES
*
PATHAREA IOSDPATH
END IOSCDRE
Chapter 89. IOSCHPD — IOS CHPID description service

Description

The IOSCHPD macro returns the acronym, description, attributes, and/or the Worldwide Port Name (WWPN) of a channel path (CHP) or channel path type.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem or Supervisor state and any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR).</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks may be held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space or be in an address/data space that is addressable through a public entry on the callers dispatchable unit access list (DU-AL).</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

The parameter list must be in the caller’s primary address space or be addressable via the dispatchable unit access list.

The LINKAGE=BRANCH option is limited to callers which meet the following criteria:

- supervisor state and key 0
- 31 bit addressing mode
- primary ASC mode
- the parameter list resides in fixed or DREF storage

Input register information

Before issuing the IOSCHPD macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

The contents of registers 14 through 1 are altered during processing.

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
</tbody>
</table>
**IOSCHPD macro**

1  Unpredictable (Used as a work register by the system)
2-13  Unchanged
14  Unpredictable (Used as a work register by the system)
15  Return code

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Unpredictable (Used as work registers by the system)</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Unpredictable (Used as work registers by the system)</td>
</tr>
</tbody>
</table>

**Performance implications**

None.

**Syntax**

The IOSCHPD macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSCHPD.</td>
</tr>
<tr>
<td>IOSCHPD</td>
<td>One or more blanks must follow IOSCHPD.</td>
</tr>
<tr>
<td>CHPID=chpid</td>
<td>chpid: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_CHP_TYPE=chp_type</td>
<td>chp_type: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_CHP_PARM=chp_parm</td>
<td>chp_type: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_CHP_PARM=0</td>
<td>Default: 0</td>
</tr>
<tr>
<td>_ACRONYM=acronym</td>
<td>acronym: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_DESC=desc</td>
<td>desc: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_ATTR=attr</td>
<td>attr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_WWPN=wwpn</td>
<td>wwpn: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_ND=xnd</td>
<td>xnd: Optional 32-character output.</td>
</tr>
</tbody>
</table>
Note: To use the IOSCHPD macro, you need to specify the following parameters:
- Either CHPID or CHP_TYPE
- One or more parameters among ACRONYM, DESC, ATTR, and WWPN.

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IOSCHPD macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

CHPID=chpid
A parameter which specifies the CHPID number for which to retrieve the attributes, acronym, description, and/or WWPN.

If the CHPID is defined as a managed channel path, the description and acronym returned will indicate that the channel path is managed. Otherwise, a non-managed description and acronym will be returned.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

CHP_TYPE=chp_type
A parameter which specifies the channel path type for which to retrieve the attributes, acronym, description, and/or WWPN. The channel path type can be
obtained by invoking the UCBINFO PATHINFO macro and mapping the results with the IOSDPATH mapping macro. (The interface type is in the field called PathIntType).

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

**CHP_PARM=CHP_parm**

An optional input parameter, used only with CHP_TYPE=chp_type parameter, that specifies the channel path parameter. A value of 1 is the managed option and 0 (the default) is the non-managed option. If 1 is specified, and if the CHP type is managed, the description and acronym returned will indicate that the CHP type is managed.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

**ATTR=attr**

An optional parameter, used only with the CHPID parameter, that designates the output area that is to receive the CHPID attributes. The attributes are mapped by mapping macro IOSDCHPD.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

**ACRONYM=acronym**

An optional parameter that designates the output area that is to receive the acronym.

To code: Specify the RS-type address, or address in register (2)-(12), of a 5-character field.

**DESC=desc**

An optional parameter that designates the output area that is to receive the description.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

**WWPN=wwpn**

An optional parameter, used only with the CHPID parameter, that designates the output area that is to receive the Worldwide Port Name (WWPN). (If the WWPN is not available, zeroes will be returned.)

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

**ND=xnd**

An optional parameter that designates the output area that is to receive the node descriptor for the channel.

**LINKAGE=SYSTEM**

**LINKAGE=BRANCH**

An optional parameter that indicates whether a branch-entry linkage should be generated or a Program Call should be issued for the routine invocation. The default is LINKAGE=SYSTEM.

**LINKAGE=SYSTEM**

requests Program Call invocation.

**LINKAGE=BRANCH**

requests branch-entry invocation. The LINKAGE=BRANCH option is
intended for performance-sensitive invokers or programs that require this function during NIP before a PC can be issued. See RESTRICTIONS for the restrictions on branch-entry invocation.

\texttt{RETCODE=retcode}
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

\texttt{RSNCODE=rsncode}
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

\texttt{PLISTVER=IMPLIED_VERSION}
\texttt{PLISTVER=MAX}
\texttt{PLISTVER=1}
\texttt{PLISTVER=2}
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
  
  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **1**, which supports all parameters except those specifically referenced in higher versions.
- **2**, which supports ATTR and WWPN, in addition to those from version 1.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1 or 2

\texttt{MF=S}
\texttt{MF=(L, list addr)}
\texttt{MF=(L, list addr, attr)}
\texttt{MF=(L, list addr, 0D)}
\texttt{MF=(E, list addr)}
\texttt{MF=(E, list addr, COMPLETE)}
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.
IOSCHPD macro

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,,list addr
   The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,,attr
   An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

,,COMPLETE
   Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

When the IOSCHPD macro returns control to your program:
- GPR 15 (and re code, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNCODE) reason code.

The following table identifies the hexadecimal return and reason codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Reason Codes, Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>The acronym and/or description has been returned.</td>
</tr>
</tbody>
</table>
## Table 83. Return and Reason Codes for the IOSCHPD Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Reason Codes, Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>The acronym and/or description have not been returned (the acronym and description output areas have been set to zeroes).</td>
</tr>
<tr>
<td></td>
<td><strong>Reason Code</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td>00</td>
<td>The system could not determine the CHP type from the input CHPID.</td>
</tr>
<tr>
<td>01</td>
<td>The input CHPID is not configured.</td>
</tr>
<tr>
<td>02</td>
<td>The CHP type obtained from the input CHPID is not valid.</td>
</tr>
<tr>
<td>03</td>
<td>The input CHP type is invalid.</td>
</tr>
<tr>
<td>04</td>
<td>The input CHP_PARM is invalid.</td>
</tr>
<tr>
<td>05</td>
<td>The managed option (1) was specified for the CHP_PARM, but the CHP type is one that does not support dynamic channel path management. The default acronym and/or description is returned.</td>
</tr>
<tr>
<td>08</td>
<td>Error in caller’s parameters.</td>
</tr>
<tr>
<td></td>
<td><strong>Reason Code</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td>01</td>
<td>The caller specified an invalid ALET.</td>
</tr>
<tr>
<td>02</td>
<td>An error occurred in accessing the caller’s parameter list.</td>
</tr>
<tr>
<td>03</td>
<td>The ATTR= keyword can only be specified with CHPID=.</td>
</tr>
<tr>
<td>0C</td>
<td>Recovery was entered.</td>
</tr>
<tr>
<td>20</td>
<td>Recovery was entered.</td>
</tr>
</tbody>
</table>
IOSCHPD macro
Chapter 90. IOSCMB — Locate the channel measurement block (CMB)

Description

The IOSCMB macro locates the channel measurement block (CMB) for a UCB and returns the data in either a 32 byte CMB format or a 64 byte ECMB format. This service eliminates the need for programs to know the format and location of the CMB.

Environment

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state, zero PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts. If the caller is disabled, the parameter list (including any data areas pointed to from the parameter list) must be in fixed or DREF storage.</td>
</tr>
</tbody>
</table>

Locks: The caller is not required to hold any locks on entry.
Control parameters: Must be in the primary address space

Programming requirements

None.

Restrictions

If the invoker is disabled, the parameter list, which includes any data areas pointed to by the parameter list, must reside in fixed or DREF storage.

Input register information

Before issuing the IOSCMB macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a 36-word save area</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>
Return code

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>

Performance implications
None.

Syntax
The IOSCMB macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSCMB.</td>
</tr>
<tr>
<td>IOSCMB</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSCMB.</td>
</tr>
<tr>
<td>GET</td>
<td>Default: GET</td>
</tr>
<tr>
<td>_UCBPTR=ucbptr addr</td>
<td>ucbptr addr: Symbol, RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>_CMBAREA=cmbarea addr</td>
<td>cmbarea addr: Symbol, RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>_CMBLEN=64</td>
<td>Default: CMBLEN=64</td>
</tr>
<tr>
<td>_CMBLEN=32</td>
<td></td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained below:

GET
Requests that the system locates the channel measurement block (CMB) for a UCB and return the data in either the old CMB format or the new ECMB format.

_UCBPTR ucbptr addr
Specifies a fullword containing the address of the UCB common segment whose CMB is to be returned.
\texttt{CMBAREA \textit{cmbarea addr}}

Specifies the address of a area to hold the measurement block being returned. The area can be either 32–bytes or 64–bytes, depending on what you specify for CMLEN.

\texttt{, CMLEN=64}

\texttt{, CMLEN=32}

Specifies whether the area pointed to by CMBAREA is:

- 64 bytes and the channel measurement block info is to be returned in ECMB format, mapped by IRAECMB.
- 32 bytes and the channel measurement block information is to be returned in CMB format, mapped by IRACMB

**Return and reason codes**

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>-</td>
<td>Meaning: Successful completion of the IOSCMB request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>08</td>
<td>01</td>
<td>Meaning: The IOSCMB request could not complete. No CMB was assigned for the device.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None; do not reissue this macro.</td>
</tr>
</tbody>
</table>

**Example**

None.
Chapter 91. IOSCMXA — Obtain address of the UCB common extension segment

Note: The UCBLOOK macro is the preferred programming interface.

Description

The IOSCMXA macro obtains the address of the UCB common extension segment. To map the UCB common extension segment, use the UCBCMEXT DSECT of the IEFUCBOB mapping macro.

Note: If you input a captured UCB address, you receive the address of the captured UCB common extension segment.

The IOSCMXA macro provides faster performance than the UCBLOOK macro; however, if the caller uses UCBLOOK to obtain several addresses in the same invocation, UCBLOOK might provide better performance than an IOSCMXA macro and an IOSUPFA macro. The UCBLOOK macro also validates input parameters and provides recovery. However, UCBLOOK cannot be used to obtain a captured UCB common extension address because UCBLOOK returns only actual UCB addresses.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state and any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>The caller may hold locks, but is not required to hold any.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>The input parameter must be in the primary address space. If the caller is disabled, the parameter list must reside in fixed or disabled reference (DREF) storage.</td>
</tr>
</tbody>
</table>

Programming requirements

The caller must pass a valid captured or actual UCB address.

The caller must pin the UCB or otherwise guarantee that the UCB will not be deleted. (If the caller issues a UCBLOOK macro with the PIN parameter to pin the UCB, use the UCBLOOK UCBCXPTR parameter rather than the IOSCMXA macro.)

The caller must supply recovery to handle any unexpected errors, such as abends.

Restrictions

None.
### Input register information
Before issuing the IOSCMXA macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

### Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if the return code is 08</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Return address</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

### Performance implications
None.

### Syntax
The standard form of the IOSCMXA macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>UCBPtr=ucbptr addr</code></td>
<td><code>ucbptr addr</code>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>UCBCXPtr=ucbcxptr addr</code></td>
<td><code>ucbcxptr addr</code>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>RETCODE=retcode</code></td>
<td><code>retcode</code>: RS-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

\textbf{UCBPTR=ucbptr addr}

Specifies the address of a fullword field that contains the address of the UCB common segment. This address must be for the UCB, and not for a copy of the UCB.

\textbf{UCBCXPTR=ucbcxptr addr}

Specifies the address of a fullword field in which the system returns the address of the UCB common extension segment. Use the UCBCMEXT DSECT of the IEFUCBOB mapping macro to map the UCB common extension segment.

\textbf{RETCODE=retcode}

An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) — (12).

\textbf{RSNCODE=rsncode}

An optional output parameter into which the reason code is copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2) — (12).

ABEND codes

None.

Return and reason codes

When the IOSCMXA macro returns control to your program:

- GPR 15 (and retcode, when you code RETCODE) contains a return code
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNCODE) contains a reason code.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td>Meaning: Successful completion.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>08 03</td>
<td>03</td>
<td>Meaning: Program error. The UCB address provided by the caller parameter does not represent a valid UCB.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Correct the UCB address and reissue the macro.</td>
</tr>
</tbody>
</table>
IOSCMXA macro

IOSCMXA - List form

Use the list form of the IOSCMXA macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros.

Syntax

The list form of the IOSCMXA macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSCMXA.</td>
</tr>
<tr>
<td>IOSCMXA</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSCMXA.</td>
</tr>
<tr>
<td>MF=(L,list addr)</td>
<td><em>list addr</em>: symbol.</td>
</tr>
<tr>
<td>MF=(L,list addr,attr)</td>
<td><em>attr</em>: 1- to 60-character input string.</td>
</tr>
<tr>
<td>MF=(L,list addr,0D)</td>
<td>Default: 0D</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained under the standard form of the IOSCMXA macro with the following exception:

- **MF=(L,list addr)**
- **MF=(L,list addr,attr)**
- **MF=(L,list addr,0D)**

Specifies the list form of the IOSCMXA macro.

- *list addr* is the name of a storage area to contain the parameters.

- *attr* is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

IOSCMXA - Execute form

Use the execute form of the IOSCMXA macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.
### Syntax

The execute form of the IOSCMXA macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSCMXA.</td>
</tr>
<tr>
<td>IOSCMXA</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSCMXA.</td>
</tr>
<tr>
<td>UCBPTR=ucbptr addr</td>
<td><em>ucbptr addr</em>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>UCBCXPTR=ucbcxptr addr</td>
<td><em>ucbcxptr addr</em>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>MF=(E,list addr)</td>
<td><em>list addr</em>: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>MF=(E,list addr,COMPLETE)</td>
<td><strong>Default</strong>: COMPLETE</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the IOSCMXA macro with the following exception:

- **MF=(E,list addr)**
- **MF=(E,list addr,COMPLETE)**
  
  Specifies the execute form of the IOSCMXA macro.

  *list addr* specifies the area that the system uses to contain the parameters.

  COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.
IOSCMXA macro
Chapter 92. IOSCMXR — Obtain address of the UCB common extension segment

Description

Use the IOSCMXR macro to obtain the address of the unit control block (UCB) common extension segment. To map the UCB common extension segment, use the UCBCMEXT DSECT of the IEFUCBOB mapping macro.

Note: If you supply a captured UCB as input, you receive the address of the captured UCB common extension segment.

UCBLOOK and IOSCMXA macros also provide this function. However, IOSCMXR provides an alternative for passing parameters (that is, in general purpose register (GPR) 1 rather than in a parameter list). Also, UCBLOOK returns only actual, not captured, UCB addresses. For guidance about obtaining UCB information, see MVS Programming: Authorized Assembler Services Guide.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>The caller may hold locks, but is not required to hold any.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>None.</td>
</tr>
</tbody>
</table>

Programming requirements

The caller must pass a valid captured or actual UCB address.

The caller must pin the UCB or otherwise guarantee that the UCB will not be dynamically deleted.

The caller must supply recovery to handle any unexpected errors, such as abends.

Restrictions

If you input a captured UCB address, the UCB must be captured in the primary address space.

Input register information

Before issuing the IOSCMXR macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of UCB common segment of the UCB</td>
</tr>
</tbody>
</table>
Before issuing the IOSCMXR macro, the caller does not have to place any information into any access register (AR).

**Output register information**

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>Address of the UCB common extension</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**Syntax**

The standard form of the IOSCMXR macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSCMXR.</td>
</tr>
<tr>
<td>IOSCMXR</td>
<td>One or more blanks must follow IOSCMXR.</td>
</tr>
<tr>
<td>MF=(S)</td>
<td>Default: S</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

MF=(S)

Specifies the standard form of the macro. This parameter is optional.
ABEND codes
None.

Return and reason codes
None.
Chapter 93. IOSCUINF — Control unit information service

Description
The IOSCUINF macro provides data of the specific control unit according to requests and also gives user the ability to reset high watermark measurements.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem or Supervisor state. Any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB mode</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
</tbody>
</table>

Programming requirements
None.

Restrictions
- Callers cannot hold any locks that prevent the IOSCUINF service from obtaining the IOSYNCH lock.
- The LINKAGE=BRANCH option is limited to callers that meet all of the following criteria:
  - supervisor state key 0
  - 31-bit addressing mode
  - Primary ASC mode
  - The parameter list resides in fixed or DREF storage
- No information is returned either by a control unit number specified for the CU keyword or by a token NED specified for the TOKENNED keyword in the CTC device.

Input register information
Before issuing the IOSCUINF macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
The contents of registers 14 through 1 are altered during processing.

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>
IOSCUINF macro

2-13  Unchanged
14    Used as a work register by the system
15    Return code

Performance implications
None.

Syntax
The standard form of the IOSCUINF macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSCUINF.</td>
</tr>
<tr>
<td>IOSCUINF</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSCUINF.</td>
</tr>
<tr>
<td>[xlabel]</td>
<td>An optional symbol, starting in column 1, that is the name on the IOSCUINF macro invocation. DEFAULT: No name</td>
</tr>
<tr>
<td>CU=cu</td>
<td>cu: The name (RS-type) of a halfword input.</td>
</tr>
<tr>
<td>,TOKENNED=tokenned</td>
<td>tokenned: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,CLASS</td>
<td></td>
</tr>
<tr>
<td>,CUCLASS=ALL</td>
<td>TAPE</td>
</tr>
<tr>
<td>,GROUP</td>
<td></td>
</tr>
<tr>
<td>,CUGROUP=PAV</td>
<td>HYPERPAV] Default: PAV Note: Specify only one of the above keywords: CU, TOKENNED, CLASS, or GROUP.</td>
</tr>
<tr>
<td>,OUTPUT_AREA=output_area</td>
<td>output_area: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RESET_MEASURES]</td>
<td>Default: None</td>
</tr>
<tr>
<td>[PATHINFO]</td>
<td>Default: None</td>
</tr>
<tr>
<td>,LINKAGE=SYSTEM</td>
<td>BRANCH] Default: SYSTEM</td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[.RETCODE=retcode]</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[.RSNCODE=rsnco]d</td>
<td>rsnco: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[.PLISTVER=plistver</td>
<td>IMPLIED_VERSION]</td>
</tr>
<tr>
<td>[.MF=S]</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>[.MF=(L,mfctrl,mfattr, 0D)]</td>
<td></td>
</tr>
<tr>
<td>[.MF=(E,mfctrl,COMPLETE)]</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

In the following set of mutually exclusive keywords, only one keyword must be specified.

**CU=**
cu
The name (RS-type) of a halfword input that contains the number of the physical control unit that the data is to be retrieved from.

**TOKENED=**
tokenned
The name (RS-type) of a 32-character input of the token NED that is the worldwide-unique identifier for the subsystem to which information is to be returned.

**CLASS**
Indicates that a control unit class is specified.

**CUCLASS=**

An optional keyword input that specifies a control unit class for which the data is to be retrieved.

Default: ALL.

**CUCLASS=ALL**
Requests data for all control units in the I/O configuration except for those in the CTC device class.

**CUCLASS=TAPE**
Requests data for TAPE device class.

**CUCLASS=COMM**
Requests data for communications device class.

**CUCLASS=DASD**
Requests data for DASD device class.

**CUCLASS=DISP**
Requests data for display device class.

**CUCLASS=UREC**
Requests data for unit record device class.

**CUCLASS=CHAR**
Requests data for character reader device class.
IOSCUINF macro

GROUP
Indicates that a group is specified.

CUGROUP=PAV | HYPERPAV
An optional keyword input that specifies a control unit group for
which the data is to be retrieved.

DEFAULT: PAV.
CUGROUP=PAV
Requests data for parallel access volume (PAV) control units.
CUGROUP=HYPERPAV
Requests data for hyper parallel access volume (HYPERPAV)
control units.

End of the mutually exclusive keywords.

OUTPUT_AREA=output_area
A required pointer output that contains the address of the requested data. The
data is mapped by IOSDCUIN. The OUTPUT_AREA is obtained by the service
and must be released by the caller.

To code: Specify the RS-type address of a fullword field, or register (2) - (12)
(ASM only).

RESET_MEASURES
Indicates that the high watermarks are to be reset for those control units that
the data was collected for.

DEFAULT: NONE.

PATHINFO
Indicates that path information is to be returned for the control units for which
data was collected. The path information includes the CU number, the interface
id, the tag, the CHPID, the link address, and the WWPN for each path
attached to the control unit.

DEFAULT: NONE.

LINKAGE=SYSTEM | BRANCH
An optional keyword input that indicates whether a program call is issued or a
branch-entry linkage is generated for the routine invocation.

DEFAULT: SYSTEM.
LINKAGE=SYSTEM
Requests program call invocation.
LINKAGE=BRANCH
Requests branch-entry invocation. See "Restrictions" on page 979 for
the restrictions on branch-entry invocation.

,RETCODE=retcode
An optional output parameter into which the return code is to be copied from
GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12)
(ASM only).

,RSCODE=rscode
An optional output parameter into which the reason code is to be copied from
GPR 0.
To code: Specify the RS-type address of a fullword field, or address in register (2) - (12) (ASM only).

\texttt{,PLISTVER=plistver|IMPLIED_VERSION}

An optional byte input decimal value in the "1-1" range that specifies the macro version. PLISTVER is the only parameter allowed on the list form of MF. This parameter determines which parameter list the system generates.

PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values can be:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM suggests that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **1**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- 1

\texttt{,MF=S}
\texttt{,MF=(L,list addr)}
\texttt{,MF=(L,list addr,attr )}
\texttt{,MF=(L,list addr,OD )}
\texttt{,MF=(E,list addr)}
\texttt{MF=(E,list addr,COMPLETE)}

An optional keyword input that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

\texttt{,list addr}

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1) - (12).

\texttt{,attr}

An optional 1-60-character input string that you use to force boundary
IOSCUINF macro

alignment of the parameter list. Use a value of 0F to force the parameter
list to a word boundary, or 0D to force the parameter list to a doubleword
boundary. If you do not code attr, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply
defaults for omitted optional parameters.

ABEND codes
None.

Return and reason codes
The following table contains hexadecimal return and reason codes, the equate
symbols associated with each reason code, and the meaning for each return and
reason code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None</td>
<td>Explanation: IOSCUINF request successful.</td>
</tr>
<tr>
<td>04</td>
<td>None</td>
<td>Explanation: Find no control units that match the requested criteria.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0001</td>
<td>Explanation: Can not use this service in AR ASC mode. Note: The OUTPUT_AREA was not returned by the service and should not be released by the caller.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0002</td>
<td>Explanation: The selection code is not valid. Note: The OUTPUT_AREA was not returned by the service and should not be released by the caller.</td>
</tr>
<tr>
<td>20</td>
<td>None</td>
<td>Explanation: An unexpected error occurred. Note: The OUTPUT_AREA was not returned by the service and should not be released by the caller.</td>
</tr>
</tbody>
</table>
Chapter 94. IOSCUMOD — IOS control unit entry build service

Description

IOSCUMOD is a prototype module, to be used by manufacturers for creating an IOSTnnn load module and for building the control unit model table.

Programming requirements

On the first invocation of the IOSCUMOD macro, it includes the parameters listed below in the manufacturer’s module.

Restrictions

None.

Performance implications

None.

Syntax

The IOSCUMOD macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSCUMOD.</td>
</tr>
<tr>
<td>IOSCUMOD</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSCUMOD.</td>
</tr>
<tr>
<td>MANF=chpid</td>
<td>manf: Symbol up to 3 characters long.</td>
</tr>
<tr>
<td>,DEVT=devt</td>
<td>devt: Symbol up to 6 characters long.</td>
</tr>
<tr>
<td>,MODN=devt</td>
<td>modn: Symbol up to 3 characters long.</td>
</tr>
<tr>
<td>,MASK1=mask1</td>
<td>mask1: 2-byte hex symbol.</td>
</tr>
<tr>
<td>,MASK2=mask2</td>
<td>mask2: 2-byte hex symbol.</td>
</tr>
<tr>
<td>,MASK3=mask3</td>
<td>mask3: 2-byte hex symbol.</td>
</tr>
<tr>
<td>,MASK4=mask4</td>
<td>mask4: 2-byte hex symbol.</td>
</tr>
</tbody>
</table>
### Parameters

The parameters are explained as follows:

- **name**
  An optional symbol, starting in column 1, that is the name on the IOSCUMOD macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

- **MANF=manf**
  Manufacturer ID that was provided with the node descriptor.

- **DEVT=devt**
  Device type ID that was provided with the node descriptor. If a 4-character device type is entered, the two leading fields will be set to blanks.

- **MODN=modn**
  Model number ID that was provided with the node descriptor. If NULL, then the model field will be set to all blanks. Otherwise, leading zeroes must be coded.

- **MASK1=mask1**
- **MASK2=mask2**
- **MASK3=mask3**
- **MASK4=mask4**
  Hex equivalent of the masks defined. 4 hex digits must be provided.

The tag field of the node descriptor uniquely identifies the power/service boundaries of most control units. Although this is true in most cases, it is not architected that way, and different control units represent this information in different ways.

In order to be able to interpret a control units tag, each control unit will provide four 2-byte masks.

Each 2 byte mask will be ANDed against the tag field of the control unit's Node Descriptor to extract a unique indicator of the different service boundary in the control unit. The first (high order) mask will indicate the most significant single point of failure to avoid (For example, Cluster), the second mask will indicate the most significant single failure to avoid (e.g. I/O bay), and so on until the fourth mask.

There is no requirement for the masks to represent specific components of the control (e.g. Cluster vs. I/O Bay vs. Port card). The only requirement is that the masks are ordered from the most significant point of failure to least. If not all four masks are significant, they should be set to binary zeros and must be the last mask(s) of the four.

- **DCM_SUPPORTED=YES**
- **DCM_SUPPORTED=NO**
  Indicates that the control unit does or does not support dynamic channel path management. Control units which support ESCON interfaces and are completely non-synchronous should be capable of being supported by DCM. Control units which transfer data synchronously from the media, or remain...
**IOSCUMOD macro**

Connected to the channel while waiting for data to transfer between the media and the cache (or channel), are not supported. The default is YES.

**ABEND codes**

None.

**Return and reason codes**

None.

System macros require High Level Assembler. Assembler language programming is described in the following information:

- [HLASM Programmer's Guide](#)
- [HLASM Language Reference](#)

Using this information also requires you to be familiar with the operating system and the services that programs running under it can invoke.
Chapter 95. IOSDCXR — Obtain address of the device class extension segment

Description

Use the IOSDCXR macro to obtain the address of the unit control block (UCB) device class extension segment. For example, the DASD device class extension segment is mapped by the IECDDCE macro and the tape device class extension segment is mapped by the IECUCBCX macro.

Note: If you supply a captured UCB as input, you receive the address of the captured UCB device class extension segment except under either of the following conditions:

- The unit information module (UIM) indicates that the device class extension segment can reside above 16 megabytes independent of the rest of the UCB.
- The UIM indicates that a single device class extension segment can be shared by multiple UCBs.

In these cases, you receive the address of the actual, not captured, UCB device class extension segment.

Other macros provide addresses to other UCB segments. For example, UCBLOOK, IOSCMXA, and IOSCMXR provide the address of the UCB common extension segment. For guidance about obtaining UCB information, see z/OS MVS Programming: Authorized Assembler Services Guide.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state and any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled or disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>None.</td>
</tr>
</tbody>
</table>

Programming requirements

The caller must pass a valid captured or actual UCB address.

The caller must pin the UCB or otherwise guarantee that the UCB will not be dynamically deleted.

The caller must supply recovery to handle any unexpected errors, such as abends.

Restrictions

If you input a captured UCB address, the UCB must be captured in the primary address space.
**Input register information**

Before issuing the IOSDCXR macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of UCB common segment</td>
</tr>
</tbody>
</table>

Before issuing the IOSDCXR macro, the caller does not have to place any information into any access register (AR).

**Output register information**

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>Address of the UCB device class extension segment if the UCB has a device class extension segment. Zero if the UCB does not have a device class extension segment.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**Syntax**

The standard form of the IOSDCXR macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSDCXR.</td>
</tr>
<tr>
<td>IOSDCXR</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSDCXR.</td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

\texttt{MF=(S)}

Specifies the standard form of the macro. This parameter is optional.

ABEND codes

None.

Return and reason codes

None.
Chapter 96. IOSENQ — IOS ENQ service

Description

IOSENQ allows you to perform ENQs and DEQs on certain I/O Supervisor (IOS) resources. Currently, the following functions can be serialized:

- Dynamic channel path management
- Dynamic I/O processing

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state. Zero PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks may be held.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

The caller should include the IOSDENQ macro to get equate symbols for the return and reason codes.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

The caller must not have a pending ENQ for the same resource managed by this service.

Input register information

Before issuing the IOSENQ macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

The contents of registers 14 through 1 are altered during processing.

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR15 is not 0</td>
</tr>
<tr>
<td>1</td>
<td>Unpredictable (Used as a work register by the system)</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Unpredictable (Used as a work register by the system)</td>
</tr>
</tbody>
</table>
IOSENQ macro

15 Return code

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Unpredictable (Used as work registers by the system)</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Unpredictable (Used as work registers by the system)</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**Syntax**

The IOSENQ macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSENQ.</td>
</tr>
<tr>
<td>IOSENQ</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSENQ.</td>
</tr>
<tr>
<td>RESOURCE=DYNCHPID</td>
<td></td>
</tr>
<tr>
<td>,REQUEST=ENQ</td>
<td></td>
</tr>
<tr>
<td>,STATE=SHARED</td>
<td></td>
</tr>
<tr>
<td>,STATE=EXCLUSIVE</td>
<td></td>
</tr>
<tr>
<td>,COND=NO</td>
<td></td>
</tr>
<tr>
<td>,COND=YES</td>
<td></td>
</tr>
<tr>
<td>,WAITTIME=waittime</td>
<td>Required choice with REQUEST=ENQ,COND=YES.</td>
</tr>
<tr>
<td>,WAITTIME=SYSTEM_DEFINED</td>
<td>Default: SYSTEM_DEFINED.</td>
</tr>
<tr>
<td>,REQUEST=DEQ</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
### Parameters

The parameters are explained as follows:

**name**

An optional symbol, starting in column 1, that is the name on the IOSENQ macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

**RESOURCE=DYNCHPID**

**RESOURCE=NOT_SUPPORTED**

A required input parameter. Indicates that this request will deal with the dynamic channel path management ENQ resource.

**REQUEST=ENQ**

**REQUEST=DEQ**

A required input parameter.

**REQUEST=ENQ**

Indicates that the request is to perform an ENQ operation.

**STATE=SHARED**

**STATE=EXCLUSIVE**

A required input parameter if REQUEST=ENQ is specified.

**STATE=SHARED**

Indicates that the ENQ should be obtained in shared state.

**STATE=EXCLUSIVE**

Indicates that the ENQ should be obtained in exclusive state.

**COND=NO**

**COND=YES**

A required input parameter if REQUEST=ENQ is specified.

**COND=NO**

Indicates that this is not a conditional ENQ. Control will only be returned to the caller when the ENQ is held.

**COND=YES**

Indicates that this is a conditional ENQ. If the ENQ cannot be obtained within the given length of time, processing is ended, and a return code indicating this situation is provided to the caller.
IOSENQ macro

\[ \text{,WAITTIME} = \text{waittime} \]
\[ \text{,WAITTIME=SYSTEM_DEFINED} \]
A required input parameter if REQUEST=ENQ,COND=YES is specified.

\[ \text{,WAITTIME} = \text{waittime} \]
The name (RS-type), or address in register (2)-(12), of a fullword input that specifies the maximum time in hundredths of seconds that the system is to wait for the ENQ to be obtained. A value of 0, or omitting this parameter, results in the system using a pre-determined wait time. The value is treated as a 32-bit unsigned number.

\[ \text{,WAITTIME=SYSTEM_DEFINED} \]
The pre-determined default time in hundredths of seconds that the system is to wait for the ENQ to be obtained.

\[ \text{,REQUEST=DEQ} \]
Indicates that the request is to perform a DEQ operation. A warning return code will result if the ENQ is not held.

\[ \text{,RETCODE} = \text{retcode} \]
An optional output parameter into which the return code is to be copied from GPR 15.

to code: Specify the RS-type address of a fullword field, or register (2)-(12).

\[ \text{,RSNCODE=rscnode} \]
An optional output parameter into which the reason code is to be copied from GPR 0.

to code: Specify the RS-type address of a fullword field, or register (2)-(12).

\[ \text{,PLISTVER=IMPLIED_VERSION} \]
\[ \text{,PLISTVER=MAX} \]
\[ \text{,PLISTVER=1} \]
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

* IMPLIED_VERSION, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

* MAX, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

* 1, if you use the currently available parameters.

to code: Specify one of the following:

* IMPLIED_VERSION
* MAX
* A decimal value of 1
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

list addr

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

attr

An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

COMPLETE

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

**ABEND codes**

The caller may get the following abend codes:

<table>
<thead>
<tr>
<th>ABEND Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C4-4</td>
<td>Your program was not in supervisor state with PSW key 0.</td>
<td>Call IOSENQ only when in supervisor state with PSW key 0.</td>
</tr>
</tbody>
</table>

| B78-8      | Your program was in problem state with PSW key 8-15. | Call IOSENQ only when in supervisor state with PSW key 0. |

**Return and reason codes**

Macro IOSDENQ provides equate symbols for the return and reason codes.

When the IOSENQ macro returns control to your program:

- GPR 15 (and retcode, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNCODE) contains a reason code.
The following table identifies the hexadecimal return and reason codes:

Table 87. Return and Reason Codes for the IOSENQ Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Reason Codes, Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Equate Symbol: IOSENQRc_OK</td>
</tr>
<tr>
<td></td>
<td>ENQ The ENQ is held.</td>
</tr>
<tr>
<td></td>
<td>DEQ The DEQ is held.</td>
</tr>
<tr>
<td>04</td>
<td>Equate Symbol: IOSENQRc_Warn</td>
</tr>
<tr>
<td></td>
<td>Reason Code</td>
</tr>
<tr>
<td></td>
<td>Meaning/Action</td>
</tr>
<tr>
<td>01</td>
<td>Equate Symbol: IOSENQRsnEnqAlreadyHeld</td>
</tr>
<tr>
<td></td>
<td>Meaning: For ENQ, this task already holds the ENQ. It could be held either in shared or exclusive state.</td>
</tr>
<tr>
<td></td>
<td>Action: Avoid using the IOSENQ REQUEST=ENQ function when you already hold the requested ENQ.</td>
</tr>
<tr>
<td>02</td>
<td>Equate Symbol: IOSENQRsnEnqNotHeld</td>
</tr>
<tr>
<td></td>
<td>Meaning: DEQ was requested but the caller did not have the ENQ.</td>
</tr>
<tr>
<td></td>
<td>Action: None required.</td>
</tr>
<tr>
<td>08</td>
<td>Equate Symbol: IOSENQRc_Env</td>
</tr>
<tr>
<td></td>
<td>Reason Code</td>
</tr>
<tr>
<td></td>
<td>Meaning/Action</td>
</tr>
<tr>
<td>01</td>
<td>Equate Symbol: IOSENQRsnBadRequest</td>
</tr>
<tr>
<td></td>
<td>Meaning: An incorrect request was specified.</td>
</tr>
<tr>
<td></td>
<td>Action: Check for storage overlays.</td>
</tr>
<tr>
<td>02</td>
<td>Equate Symbol: IOSENQRsnBadResource</td>
</tr>
<tr>
<td></td>
<td>Meaning: An incorrect resource was requested.</td>
</tr>
<tr>
<td></td>
<td>Action: Check for storage overlays.</td>
</tr>
<tr>
<td>0C</td>
<td>Equate Symbol: IOSENQRc_Env</td>
</tr>
<tr>
<td></td>
<td>Reason Code</td>
</tr>
<tr>
<td></td>
<td>Meaning/Action</td>
</tr>
<tr>
<td>01</td>
<td>Equate Symbol: IOSENQRsnCouldNotGetENQ</td>
</tr>
<tr>
<td></td>
<td>Meaning: On a conditional ENQ request, the ENQ could not be obtained within the specified time period.</td>
</tr>
<tr>
<td></td>
<td>Action: Specify a longer time period, or request the ENQ unconditionally.</td>
</tr>
</tbody>
</table>
Chapter 97. IOSFBA — IOS fixed block architecture service

Description

IOSFBA is used to manage (allocate and unallocate) and perform I/O (read and write) to allocated fixed block architecture (FBA) devices. IOSFBA has the following functions:

ALLOCATE
Allocates one or more FBA devices to be used with the READ or WRITE IOSFBA service.

QUERY
Provides information about the requested devices. The caller provides a list of device numbers for which the service returns device attributes.

READ
Initiates read operations for one or more devices as described in the supplied device I/O list.

WRITE
Initiates write operations for one or more devices as described in the supplied device I/O list.

ERASE
Initiates erase operations for a contiguous area for one or more devices as described in the supplied device I/O list.

CLEANUP
 Cleans up resources associated with an I/O token.

UNALLOCATE
Unallocates one or more FBA devices that were previously allocated with this service.

Note: Mapping macros for the IOS fixed block architecture services are contained in IOSFBA.

Environment

The requirements for the caller are:

Environmental factor  Requirement
Minimum authorization:  Supervisor state, PSW Key 0.
Dispatchable unit mode:  Task
Cross memory mode:  PASN=HASN=SASN
AMODE:  31- or 64-bit
ASC mode:  Primary
Interrupt status:  Enabled for I/O and external interrupts.
Locks:  No locks may be held.

Programming requirements

None.
**Restrictions**

The invoker must have SAF authorization to facility 'IOSFBA'. Specifically, the invoker must have UPDATE authority to facility class 'IOSFBA'.

**Input register information**

Before issuing the IOSFBA macro, the caller must ensure that general register 13 contains the address of a 72 byte save area (for AMODE(31) callers) or 216 byte save area (for AMODE(64) callers). The save area must be in primary storage in the first 2GB of storage. The caller does not have to place any information into any other general purpose register (GPR) unless using it in register notation for a particular parameter or using it as a base register.

**Output register information**

The contents of registers 14 through 1 are altered during processing.

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Unpredictable (Used as a work register by the system)</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Unpredictable (Used as a work register by the system)</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

**Performance implications**

None.

**Syntax**

The IOSFBA macro is written as follows:

```assembly
main diagram

name IOSFBA.

ALLOCATE parameters-1.

QUERY, DEVLISTPTR = devlistptr, DEVSRCPTR = devsrcptr.

READ, parameters-2.

WRITE, parameters-3.

ERASE, DEVSRCPTR = devsrcptr.

CLEANUP, IOTOKEN = iotoken.

UNALLOCATE, DEVSRCPTR = devsrcptr.

RETCODE = retcode.

RSMCODE = rsncode.

PLISTVER = IMPLIED_VERSION.

PLISTVER = MAX.

PLISTVER = 1.
```
Parameters

The parameters are explained as follows:
name

An optional symbol, starting in column 1, that is the name on the IOSFBA macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,ACCESS=SINGLE
,ACCESS=READ
,ACCESS=WRITE
,ACCESS=ANY

When ALLOCATE is specified, ACCESS is an optional parameter that indicates the type of allocation that should be performed. The default is ACCESS=SINGLE.

,ACCESS=SINGLE

The FBA device is allocated for use by a single z/OS system. Note that one or more distributed systems can access this device in addition to the single z/OS system. Access from distributed systems is controlled through the use of LUN masking or using the persistent reserve from the distributed side.

,ACCESS=READ

The FBA device is allocated for READ only on the system where this request is received. This device can also be allocated on another z/OS system by requesting ACCESS=WRITE. Note that one or more distributed systems can access this device in addition to the pair of z/OS systems that performed the z/OS allocations. Access from distributed systems is controlled through the use of LUN masking or using the persistent reserve from the distributed side.

,ACCESS=WRITE

The FBA device is allocated for WRITE only on the system where this request is received. This device can also be allocated on another z/OS system by requesting ACCESS=READ. Note that one or more distributed systems can access this device in addition to the pair of z/OS systems that performed the z/OS allocations. Access from distributed systems is controlled through the use of LUN masking or using the persistent reserve from the distributed side.

,ACCESS=ANY

The FBA device is allocated once on each z/OS system. Note that one or more distributed systems can access this device in addition to the z/OS systems that performed the z/OS allocations. Access from distributed systems is controlled through the use of LUN masking or using the persistent reserve from the distributed side.

ALLOCATE

A required input parameter that allocates the devices as specified in the device list (FBADL DSECT in the IOSDFBA macro). This request, if successful, allocates the requested number of devices within a sysplex for the caller's use.

To code: Specify a value.

,CLEANUP

A required input parameter that cleans up resources for the input IOTOKEN. During IOSFBA READ or IOSFBA WRITE invocations, one or more IOTOKEN areas may have been used to allow for efficiency. IOSFBA CLEANUP must be invoked once for each IOTOKEN that was used during processing.

To code: Specify a value.

,DEVCOUNT=devcount
When ALLOCATE is specified, DEVCOUNT is an optional input parameter that contains the number of devices that should be allocated from the devices specified in the device list (FBADL) addressed by the DEVLISTPTR or DEVLISTPTR64 parameter. DEVCOUNT indicates the maximum number of devices to allocate. DEVCOUNT should be less than or equal to the number of devices specified in the device list. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

When ALLOCATE is specified, DEVDESCPTR is a required output parameter of the output device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro).

The device descriptor area contains the addresses of a device descriptor entry for each device successfully allocated. The device descriptor entry contains specific device information discovered by the ALLOCATE service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro). The device descriptor entry is required input for the IOSFBA READ and IOSFBA WRITE services.

When using the IOSFBA READ or IOSFBA WRITE services, the device descriptor entry is required input for each device that is read from or written to. Refer to the DEVOLISTPTR or DEVOLISTPTR64 parameter for the IOSFBA READ and/or IOSFBA WRITE service for more information.

The device descriptor address is an input parameter for the IOSFBA UNALLOCATE service. The UNALLOCATE service unallocates the devices contained in the device descriptor area.

The caller is responsible for freeing or releasing this storage after the devices have been UNALLOCATED. The subpool and length of the storage are contained in the device descriptor area. The device descriptor area must be freed using either STORAGE RELEASE or FREEMAIN macro invocation.

Note: The DEVDESCPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

When QUERY is specified, DEVDESCPTR is a required output parameter of the output device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro).

The device descriptor area contains the addresses of a device descriptor entry for each device successfully queried. The device descriptor entry contains specific device information discovered by the QUERY service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro).

The caller is responsible for freeing or releasing this storage after the devices have been queried. The subpool and length of the storage are contained in the device descriptor area. The device descriptor area must be freed using either STORAGE RELEASE or FREEMAIN macro invocation.

Note: The DEVDESCPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.
To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

\`DEVDESCPTR=devdescptr\`

When ERASE is specified, DEVDESCPTR is a required input parameter of the input device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro). The device descriptor area is obtained by the IOSFBA ALLOCATE service.

The device descriptor area contains the addresses of the device descriptor entry for each device to be unallocated. The device descriptor entry contains specific device information discovered by the ALLOCATE service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro).

The status of the erase request for each device is obtained by checking the FBADDL_EraseFailed and FBADDL_NoEraseAttempted indicators in the FBADDL for each device. If FBADDL_EraseFailed is indicated, the FBADDE_COD and FBADDE_RCOD fields contain information about the I/O failure.

Note: The DEVDESCPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

\`DEVDESCPTR=devdescptr\`

When UNALLOCATE is specified, DEVDESCPTR is a required input parameter of the input device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro). The device descriptor area is obtained by the IOSFBA ALLOCATE service.

The UNALLOCATE service unallocates the devices contained in the device descriptor area, specifically each of the device descriptor entries.

The device descriptor area contains the addresses of the device descriptor entry for each device to be unallocated. The device descriptor entry contains specific device information discovered by the ALLOCATE service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro).

The status of the unallocation request for each device is obtained by checking the return code and reason code contained in the device descriptor entry (FBADDE). The return code and reason code contain the dynamic allocation (SVC 99) return and reason code or an IOSFBA service return and reason code.

The caller is responsible for freeing or releasing this storage after the devices have been UNALLOCATED. The subpool and length of the storage are contained in the device descriptor area. The device descriptor area must be freed using either STORAGE RELEASE or FREEMAIN macro invocation.

Note: The DEVDESCPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

\`DEVDESCPTR64=devdescptr64\`

When ALLOCATE is specified, DEVDESCPTR64 is a required output parameter of the output device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro).
The device descriptor area contains the addresses of a device descriptor entry for each device successfully allocated. The device descriptor entry contains specific device information discovered during by the ALLOCATE service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro). The device descriptor entry is required input for the IOSFBA READ and IOSFBA WRITE services.

When using the IOSFBA READ or IOSFBA WRITE services, the device descriptor entry is required input for each device that is read from or written to. Refer to the DEVIOLIST parameter for the IOSFBA READ and/or IOSFBA WRITE service for more information.

The device descriptor address is an input parameter for the IOSFBA UNALLOCATE service. The UNALLOCATE service unallocates the devices contained in the device descriptor area.

The caller is responsible for freeing or releasing this storage after the devices have been UNALLOCATED. The storage area must be freed using the IARST64 service.

Note: The DEVDESCPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,DEVDESCPTR64=devdescptr64

When QUERY is specified, DEVDESCPTR64 is a required output parameter of the output device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro).

The device descriptor area contains the addresses of a device descriptor entry for each device successfully queried. The device descriptor entry contains specific device information discovered by the QUERY service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro).

The caller is responsible for freeing or releasing this storage after the devices have been queried. The subpool and length of the storage are contained in the device descriptor area. The device descriptor area must be freed using either STORAGE RELEASE or FREEMAIN macro invocation.

Note: The DEVDESCPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,DEVDESCPTR64=devdescptr64

When ERASE is specified, DEVDESCPTR64 is a required input parameter of the input device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro). The device descriptor area is obtained by the IOSFBA ALLOCATE service.

The UNALLOCATE service unallocates the devices contained in the device descriptor area, specifically each of the device descriptor entries.

The device descriptor area contains the addresses of the device descriptor entry for each device to be unallocated. The device descriptor entry contains specific device information discovered by the ALLOCATE service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro).
The status of the erase request for each device is obtained by checking the
FBADDL_EraseFailed and FBADDL_NoEraseAttempted indicators in the
FBADDL for each device. If FBADDL_EraseFailed is indicated, the
FBADDE_COD and FBADDE_RCCOD fields contain information about the I/O
failure.

**Note:** The DEVDESCPTR64 parameter is allowed only when in AMODE 64 as
indicated by the SYSSTATE macro.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an
eight-byte pointer field.

```
,DEVDESCPTR64=devdescptr64
```

When UNALLOCATE is specified, DEVDESCPTR64 is a required input
parameter of the input device descriptor area mapped by the FBADDL
(defined in the IOSDFBA macro). The device descriptor area is obtained by the
IOSFBA ALLOCATE service.

The UNALLOCATE service unallocates the devices contained in the device
descrptor area - specifically each of the device descriptor entry.

The device descriptor area contains the addresses of the device descriptor entry
for each device to be unallocated. The device descriptor entry contains specific
device information discovered by the ALLOCATE service. The device
descrptor entry is mapped by the FBADDE (defined in the IOSDFBA macro).

The status of the unallocation request for each device is obtained by checking
the return code and reason code contained in the device descriptor entry
(FBADDE). The return code and reason code contain the dynamic allocation
(SVC 99) return and reason code or an IOSFBA service return and reason code.

The caller is responsible for freeing or releasing this storage after the devices
have been UNALLOCATED. The device descriptor area must be freed using
the IARST64 service.

**Note:** The DEVDESCPTR64 parameter is allowed only when in AMODE 64 as
indicated by the SYSSTATE macro.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an
eight-byte pointer field.

```
,DEVIOLISTPTR=deviolistptr
```

When READ is specified, DEVIOLISTPTR is a required input parameter of the
input device I/O list mapped by the FBADIO (defined in the IOSDFBA
macro). The device I/O list specifies the number of devices that will participate
in the IOSFBA READ service. Additionally, the device I/O list contains a
pointer to the device I/O entry for each device (mapped by the FBADIOE,
defined in the IOSDFBA macro).

The device I/O entry includes the address of the device descriptor entry (that
was returned as part of the device descriptor area by the IOSFBA ALLOCATE
service), a count of extent entries, the addresses of each extent entry (mapped
by the FBAEE, defined in the IOSDFBA macro), and if required by the caller,
the address of a status block or area (mapped by the FBAST, defined in the
IOSDFBA macro). For a complete description, see the IOSDFBA macro.

The extent entry defines the parameters of the READ I/O operation for a given
device. It defines the starting block number on the device, the number of
blocks to transfer, and the storage address or addresses to place the
information read from the FBA device.
The status block provides the caller with status information for the I/O to each device. A status block should be obtained and initialized to zeroes by the caller for each device that will participate in the IOSFBA READ service.

For AMODE(31) callers, the storage area must be addressable in AMODE(31).

**Note:** The DEVIOLISTPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a pointer field.

\texttt{DEVIOLISTPTR=deviolistptr}

When WRITE is specified, DEVIOLISTPTR is a required input parameter of the input device I/O list mapped by the FBADIOL (defined in the IOSDFBA macro). The device I/O list specifies the number of devices that will participate in the IOSFBA READ service. Additionally, the device I/O list contains a device I/O entry for each device (mapped by the FBADIOE, defined in the IOSDFBA macro).

The device I/O entry includes the address of the device descriptor entry (that was returned as part of the device descriptor area by the IOSFBAALLOCATE service), a count of extent entries, the addresses of each extent entry (mapped by the FBAEE, defined in the IOSDFBA macro), and if required by the caller, the address of a status block or area (mapped by the FBAST, defined in the IOSDFBA macro). For a complete description, see the IOSDFBA macro.

The extent entry defines the parameters of the WRITE I/O operation for a given device. It defines the starting block number on the device, the number of blocks to transfer, and the storage address or addresses to place the information read from the FBA device.

The status block provides the caller with status information for the I/O to each device. A status block should be obtained and initialized to zeroes by the caller for each device that will participate in the IOSFBA WRITE service.

For AMODE(31) callers, the storage area must be addressable in AMODE(31).

**Note:** The DEVIOLISTPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a pointer field.

\texttt{DEVIOLISTPTR64=deviolistptr64}

When READ is specified, DEVIOLISTPTR64 is a required input parameter of the input device I/O list mapped by the FBADIOL (defined in the IOSDFBA macro). The device I/O list specifies the number of devices that will participate in the IOSFBA READ service. Additionally, the device I/O list contains a device I/O entry for each device (mapped by the FBADIOE, defined in the IOSDFBA macro).

The device I/O entry includes the address of the device descriptor entry (that was returned as part of the device descriptor area by the IOSFBAALLOCATE service), a count of extent entries, the addresses of each extent entry (mapped by the FBAEE, defined in the IOSDFBA macro), and if required by the caller, the address of a status block or area (mapped by the FBAST, defined in the IOSDFBA macro). For a complete description, see the IOSDFBA macro.
The extent entry defines the parameters of the READ I/O operation for a given device. It defines the starting block number on the device, the number of blocks to transfer, and the storage address or addresses to place the information read from the FBA device.

The status block provides the caller with status information for the I/O to each device. A status block should be obtained and initialized to zeroes by the caller for each device that will participate in the IOSFBA READ service.

**Note:** The DEVIOLISTPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,DEVIOLISTPTR64=deviolistptr64
When WRITE is specified, DEVIOLISTPTR64 is a required input parameter of the input device I/O list mapped by the FBADIOL (defined in the IOSDFBA macro). The device I/O list specifies the number of devices that will participate in the IOSFBA READ service. Additionally, the device I/O list contains a device I/O entry for each device (mapped by the FBADIOE, defined in the IOSDFBA macro).

The device I/O entry includes the address of the device descriptor entry (that was returned as part of the device descriptor area by the IOSFBA ALLOCATE service), a count of extent entries, the addresses of each extent entry (mapped by the FBAEE, defined in the IOSDFBA macro), and if required by the caller, the address of a status block or area (mapped by the FBAST, defined in the IOSDFBA macro). For a complete description, see the IOSDFBA macro.

The extent entry defines the parameters of the WRITE I/O operation for a given device. It defines the starting block number on the device, the number of blocks to transfer, and the storage address or addresses to place the information read from the FBA device.

The status block provides the caller with status information for the I/O to each device. A status block should be obtained and initialized to zeroes by the caller for each device that will participate in the IOSFBA WRITE service.

**Note:** The DEVIOLISTPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,DEVLISTPTR=devlistptr
When ALLOCATE is specified, DEVLISTPTR is a required input parameter of the input device list mapped by the FBADL (defined in the IOSDFBA macro). The FBADL specifies the number of devices, the device numbers, and others to allocate. (Refer to the FBADL for specific information.)

For AMODE(31) callers, the storage area must be addressable in AMODE(31).

**Note:** The DEVLISTPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,DEVLISTPTR=devlistptr
When QUERY is specified, DEVLISTPTR is a required input parameter of the
input device list mapped by the FBADL (defined in the IOSDFBA macro). The FBADL specifies the number of devices, the device numbers, and others to allocate. (Refer to the FBADL for specific information.)

For AMODE(31) callers, the storage area must be addressable in AMODE(31).

**Note:** The DEVLISPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a pointer field.

```
,DEVLISPTR64=devlistptr64
```

When ALLOCATE is specified, DEVLISPTR64 is a required input parameter of the input device list mapped by the FBADL (defined in the IOSDFBA macro). The FBADL specifies the number of devices, the device numbers, and others to allocate. (Refer to the FBADL for specific information.)

**Note:** The DEVLISTPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

```
,DEVLISPTR64=devlistptr64
```

When QUERY is specified, DEVLISTPTR64 is a required input parameter of the input device list mapped by the FBADL (defined in the IOSDFBA macro). The FBADL specifies the number of devices, the device numbers, and others to allocate. (Refer to the FBADL for specific information.)

**Note:** The DEVLISTPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

```
,ECB=ecb
,ECB=NONE
```

When READ is specified, ECB is an optional input parameter that contains the address that points to an optional ECB that is posted when all read operations are complete. If an ECB is not specified, control returns to the invoker when all read operations have completed. The default is NONE.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a pointer field.

```
,ECB=ecb
,ECB=NONE
```

When WRITE is specified, ECB is an optional input parameter that contains the address that points to an optional ECB that is posted when all write operations are complete. If an ECB is not specified, control returns to the invoker when all write operations have completed. The default is NONE.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a pointer field.

```
,ERASE
```

A required input parameter that erases a contiguous area of the device or devices as specified in the device list mapped by the FBADL DSECT (defined in the IOSDFBA macro). Erase writes null characters (X'00').

**To code:** Specify a value.
When READ is specified, IOTOKEN is an optional input parameter that contains the address that points to an optional 32-byte area used by the IOSFBA service to store addresses and lengths of storage areas that can be reused in order to avoid system overhead of obtaining these resources on each call.

The invokers of IOSFBA can use as many unique 32-byte IOTOKEN areas as desired. When IOTOKEN is used,

- The area specified by the IOTOKEN should be initially cleared by the calling program before the first usage of the IOTOKEN area.
- Each IOTOKEN is used to represent storage areas for the life of an I/O request. The caller should not reuse an IOTOKEN until the I/Os initiated for it have completed. For synchronous callers, the IOTOKEN can be immediately reused. For asynchronous callers (when an ECB is provided), the caller must not reuse an IOTOKEN until the ECB has been posted.
- IOSFBA CLEANUP must be invoked for each IOTOKEN that was used to ensure that task related storage is released.

The default is NONE.

To code: Specify the RS-type address of a pointer field.

When WRITE is specified, IOTOKEN is an optional input parameter that contains the address that points to an optional 32-byte area that is used by the IOSFBA service to store addresses and lengths of storage areas that can be reused in order to avoid system overhead of obtaining these resources on each call.

The invokers of IOSFBA can use as many unique 32-byte IOTOKEN areas as desired. When IOTOKEN is used,

- The area specified by the IOTOKEN should be initially cleared by the calling program before the first usage of the IOTOKEN area.
- Each IOTOKEN is used to represent storage areas for the life of an I/O request. The caller should not reuse an IOTOKEN until the I/Os initiated for it have completed. For synchronous callers, the IOTOKEN can be immediately reused. For asynchronous callers (when an ECB is provided), the caller must not reuse an IOTOKEN until the ECB has been posted.
- IOSFBA CLEANUP must be invoked for each IOTOKEN that was used to ensure that task related storage is released.

The default is NONE.

To code: Specify the RS-type address of a pointer field.

When CLEANUP is specified, IOTOKEN is a required input parameter that contains the address that points to a 32-byte area that is used by the IOSFBA service to store addresses and lengths of storage areas that can be reused in order to avoid system overhead of obtaining these resources on each call.

To code: Specify the RS-type address of a pointer field.
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

list addr
The name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this is an RS-type address or an address in register (1)-(12).

attr
An optional 1 to 60 character input string used to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

MINDEVCOUNT=mindevcount
MINDEVCOUNT=0
When ALLOCATE is specified, MINDEVCOUNT is an optional input parameter that indicates the minimum number of devices that must be allocated to fulfill this allocate request. The devices are specified in the device list (FBADL) addressed by the DEVLISTPTR or DEVLISTPTR64 parameter. This number should be less than or equal to the number specified in DEVCOUNT.

If the caller requests DEVCOUNT=x and MINDEVCOUNT=y, the IOSFBA service attempts to allocate the requested number of devices (as specified by the DEVCOUNT=x parameter). If 'x' devices are not available to be allocated, IOSFBA ALLOCATE service attempts to allocate as many devices that are available. The ALLOCATE request is considered successful if at least 'y' devices are allocated (as specified by the MINDEVCOUNT=y parameter). The ALLOCATE request is considered unsuccessful if 'y' devices (as specified by the MINDEVCOUNT=y parameter) are not allocated and a return code is set indicating the ALLOCATE request failed since the minimum number of devices could not be allocated.
The count or number of devices that have been allocated is contained in the device descriptor area (mapped by the FBADDL).

If this keyword is omitted or specified as 0, the MINDEVCOUNT is assumed to be the value specified on the DEVCOUNT keyword. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

\texttt{PLISTVER=IMPLIED\_VERSION}
\texttt{PLISTVER=MAX}
\texttt{PLISTVER=1}

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED\_VERSION**, the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED\_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **1**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED\_VERSION
- MAX
- A decimal value of 1

\texttt{QUERY}

A required input parameter, queries information about a list of devices. Information is returned for each device in the list of devices identified in the DEVLIST keyword.

To code: Specify a value.

\texttt{READ}

A required input parameter, reads information from one or more FBA devices as specified by the device I/O list (DEVIOLIST parameter).

To code: Specify a value.

\texttt{RESERVED}

When READ is specified, RESERVED is an optional input parameter indicating that the device or devices for this READ operation may be serialized by persistent reserve from a distributed client. When this keyword is specified, z/OS I/O operations are permitted to the devices while the persistent reserve is held. This keyword should only be used when the invoking program is coordinating I/O activity between z/OS and the distributed client that owns the persistent reserve. The default is NONE.

To code: Specify a value.
When WRITE is specified, RESERVED is an optional input parameter that indicates that the device or devices for this WRITE operation may be serialized by persistent reserve from a distributed client. When this keyword is specified, z/OS I/O operations are permitted to the devices while the persistent reserve is held. This keyword should only be used when the invoking program is coordinating I/O activity between z/OS and the distributed client that owns the persistent reserve. The default is NONE.

To code: Specify a value.

\texttt{RETCODE=retcode}

An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value is left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

\texttt{REUSECP=NO}

\texttt{REUSECP=YES}

When IOTOKEN=iotoken and READ are specified, REUSECP is an optional parameter indicating that the channel program for this operation is exactly the same as the previous channel program. On the first READ request, this keyword is ignored. On subsequent requests, this keyword indicates whether the exact same storage buffers and blocks on the disks are involved in the I/O operations. The exact same channel program is used to perform the I/O. Using this REUSECP=YES may provide some performance advantage since analyzing the extents and storage buffers is not required. The default is REUSECP=NO.

\texttt{REUSECP=NO}

Indicates that the channel program may vary from invocation to invocation and should be rebuilt on each IOSFBA request.

\texttt{REUSECP=YES}

Indicates that the channel program does not vary from invocation to invocation and that IOSFBA is instructed to start the previously built channel program without modification if one has been previously built. If no prior channel program was executed for the input IOTOKEN, a new one is built and saved for the next invocation. When REUSECP=YES is specified,

- You must use storage that is fixed for life of the IOTOKEN. Channel programs are not rebuilt to obtain the latest real storage address.
- You must make the address space non-swappable for the life of the IOTOKEN.

If either of these conditions cannot be met, you must not use REUSECP=YES.

\texttt{REUSECP=NO}

\texttt{REUSECP=YES}

When IOTOKEN=iotoken and WRITE are specified, REUSECP is an optional parameter that indicates that the channel program for this operation is exactly the same as the previous channel program. On the first WRITE request, this keyword is ignored. On subsequent requests, this keyword indicates whether the exact same storage buffers and blocks on the disks are involved in the I/O operations. The exact same channel program is used to perform the I/O. Using this REUSECP=YES may provide some performance advantage since analyzing the extents and storage buffers is not required. The default is REUSECP=NO.
Indicates that the channel program may vary from invocation to invocation and should be rebuilt on each IOSFBA request.

Indicates that the channel program does not vary from invocation to invocation and that IOSFBA is instructed to start the previously built channel program without modification if one has been previously built. If no prior channel program was executed for the input IOTOKEN, a new one is built and saved for the next invocation. When REUSECP=YES is specified,

- You must use storage that is fixed for life of the IOTOKEN. Channel programs are not rebuilt to obtain the latest real storage address.
- You must make the address space non-swappable for the life of the IOTOKEN.

If either of these conditions cannot be met, you must not use REUSECP=YES.

An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value is left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

A required input parameter that unallocates the devices as specified in the device list mapped by the FBADL DSECT (defined in the IOSDFBA macro).

To code: Specify a value.

A required input parameter that writes information to one or more FBA devices as specified by the device I/O list (DEVIOLIST parameter).

To code: Specify a value.

### ABEND codes
None.

### Return and reason codes
When the IOSFBA macro returns control to your program:
- GPR 15 (and retcode, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes:
<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>Successful operation.</td>
</tr>
<tr>
<td></td>
<td><strong>Reason Code</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning/Action</strong></td>
</tr>
<tr>
<td>X'00'</td>
<td>The operation was successful.</td>
</tr>
<tr>
<td>X'01'</td>
<td>(ALLOCATE) The requested number of devices (DEVCOUNT) were not available. However, FBADDL_COUNT contains the number of devices actually allocated for this request, which is greater than or equal to the minimum number of devices required (MINDEVCOUNT).</td>
</tr>
<tr>
<td>X'04'</td>
<td>Warning error.</td>
</tr>
<tr>
<td></td>
<td><strong>Reason Code</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning/Action</strong></td>
</tr>
<tr>
<td>X'01'</td>
<td>(QUERY) Information for one or more devices could not be obtained.</td>
</tr>
<tr>
<td>X'02'</td>
<td>(UNALLOCATE) One or more devices in the DEVLIST could not be unallocated.</td>
</tr>
<tr>
<td>X'08'</td>
<td>Error in the caller’s parameters.</td>
</tr>
<tr>
<td></td>
<td><strong>Reason Code</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning/Action</strong></td>
</tr>
<tr>
<td>X'01'</td>
<td>IOSFBA abended during parameter validation.</td>
</tr>
<tr>
<td>X'02'</td>
<td>(ALLOCATE) MINDEVCOUNT is greater than DEVCOUNT.</td>
</tr>
<tr>
<td>X'03'</td>
<td>(ALLOCATE, QUERY) The device list is not properly built.</td>
</tr>
<tr>
<td>X'04'</td>
<td>(READ, WRITE) The number of blocks identified in the READ or WRITE request does not properly equate with the amount of I/O buffers provided.</td>
</tr>
<tr>
<td>X'05'</td>
<td>(READ, WRITE) The requested extents to be READ or WRITTEN are not within the acceptable range of extents available on the device.</td>
</tr>
<tr>
<td>X'06'</td>
<td>An invalid function was specified for the IOSFBA invocation.</td>
</tr>
<tr>
<td>X'07'</td>
<td>(UNALLOCATE, ERASE) The device descriptor list is not properly built.</td>
</tr>
<tr>
<td>X'08'</td>
<td>(UNALLOCATE, READ, WRITE, ERASE) The device descriptor entry is not properly built.</td>
</tr>
<tr>
<td>X'09'</td>
<td>(UNALLOCATE, READ, WRITE, ERASE) The UCB specified in the device descriptor entry is not allocated.</td>
</tr>
<tr>
<td>X'0B'</td>
<td>(READ, WRITE) The buffers specified are not properly sized with the physical block size of the device. Buffer sizes must be multiples of the physical block size of the device.</td>
</tr>
<tr>
<td>X'0C'</td>
<td>(READ, WRITE) The device I/O list is not properly built.</td>
</tr>
<tr>
<td>X'0D'</td>
<td>(READ, WRITE) The device I/O entry is not properly built.</td>
</tr>
<tr>
<td>X'0E'</td>
<td>(READ, WRITE) The extent entry is not properly built.</td>
</tr>
<tr>
<td>X'0F'</td>
<td>(READ, WRITE) REUSECP=YES was specified, but the address space is swappable. The address space must be non-swappable when REUSEP=YES is specified.</td>
</tr>
</tbody>
</table>
### Table 88. Return and Reason Codes for the IOSFBA Macro (continued)

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0C'</td>
<td>Environmental error.</td>
</tr>
</tbody>
</table>

#### Reason Code

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'01'</td>
<td>(ALLOCATE) Not enough devices were available to satisfy the requested allocation. Devices must be online, usable, and not already allocated in order to be allocated by this service.</td>
</tr>
<tr>
<td>X'02'</td>
<td>(ALLOCATE) Not enough devices provided I/O responses that enabled IOSFBA to validate that they were usable, so IOSFBA was not able to satisfy the requested allocation. Devices must be online, usable, not already allocated, and must properly respond to I/O commands that query device information in order to be allocated by this service.</td>
</tr>
<tr>
<td>X'03'</td>
<td>(ALLOCATE, READ, WRITE) IOSFBA detected a serialization problem with one of the devices being allocated. A SYMREC record was written. Consider varying the device offline and investigate prior device usage before attempting to use the identified device again.</td>
</tr>
<tr>
<td>X'04'</td>
<td>IOSFBA service is not available.</td>
</tr>
<tr>
<td>X'05'</td>
<td>IOSFBA detected that the caller is not in task mode. IOSFBA must be invoked in task mode.</td>
</tr>
<tr>
<td>X'06'</td>
<td>IOSFBA detected that the caller is not enabled for I/O interrupts.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'10'</td>
<td>The READ or WRITE operation was not successful. Check the status blocks for information on the failed I/Os. Note that this return code is only valid for synchronous READ and WRITE operations. Asynchronous READ and WRITE operations are notified by a post code of X'10'.</td>
</tr>
<tr>
<td>X'14'</td>
<td>The invoker is not authorized to use this programming service.</td>
</tr>
</tbody>
</table>

#### Reason Code

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'01'</td>
<td>The invoker must be in PSW key 0-7 and in supervisor state.</td>
</tr>
</tbody>
</table>
Chapter 98. IOSHXBLK — Request to suspend and resume Basic HyperSwap services

Description

The IOSHXBLK macro is used by authorized applications to request Basic HyperSwap to temporarily suspend its activities. The macro should subsequently be used to allow Basic HyperSwap to resume its activities.

Note that obtaining and holding the block of Basic HyperSwap for a long period of time may cause one of Basic HyperSwap tasks to issue diagnostic abend 2E0.

Environment

The requirements for the caller of IOSHXBLK are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state. Any key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task mode</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>The caller may not hold any lock.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the IOSHXBLK macro, the caller does not have to place any information into any registers unless using it in register notation for a particular parameter or using it as a base register.

Register

<table>
<thead>
<tr>
<th>Register Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Undefined</td>
</tr>
<tr>
<td>1 Used by the service</td>
</tr>
<tr>
<td>2-13 Undefined</td>
</tr>
<tr>
<td>14-15 Used by the service</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
</table>

IOSHXBKL macro

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**Syntax**

The IOSHXBKL macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must precede IOSHXBKL.</td>
</tr>
<tr>
<td>IOSHXBKL</td>
<td></td>
</tr>
<tr>
<td><code>b</code></td>
<td>One or more blanks must follow IOSHXBKL.</td>
</tr>
<tr>
<td>BLOCK</td>
<td>Suspend Basic HyperSwap activities.</td>
</tr>
<tr>
<td><code>,NAMEPTR=nameptr</code></td>
<td><em>nameptr</em>: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td><code>,TOKENPTR=tokenptr</code></td>
<td><em>tokenptr</em>: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td>UNBLOCK</td>
<td>Resume Basic HyperSwap activities.</td>
</tr>
<tr>
<td><code>,NAMEPTR=nameptr</code></td>
<td><em>nameptr</em>: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td><code>,TOKENPTR=tokenptr</code></td>
<td><em>tokenptr</em>: RS-type address or address in register (2) - (12)</td>
</tr>
<tr>
<td><code>,FORCE=NO</code></td>
<td>Default: FORCE=NO</td>
</tr>
<tr>
<td><code>,FORCE=YES</code></td>
<td></td>
</tr>
<tr>
<td>TESTBLOCK</td>
<td>Test whether all systems in the SYSPLEX support programmatic blocking.</td>
</tr>
<tr>
<td>Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>PLISTVER=plistver</td>
<td></td>
</tr>
<tr>
<td>PLISTVER=IMPLIED_VERSION</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>MF=(L,mfctrl,mfattr,0D)</td>
<td></td>
</tr>
<tr>
<td>MF=(E,mfctrl,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

**name**

An optional symbol, starting in column 1, that is the name on the IOSHXBLK macro invocation. The name must conform to the rules for an ordinary assembler language symbol. The default is no name.

**BLOCK**, **UNBLOCK** and **TESTBLOCK** are mutually exclusive keys. This set is required, but only one key may be specified.

**BLOCK**

Suspend Basic HyperSwap activities.

\[\text{NAMEPTR}=\text{nameptr}\]

The name (RS-type), or address in register (2)-(12), of a required 4-byte input that contains the address of a 8-character area that contains the name of the application issuing the BLOCK request.

\[\text{TOKENPTR}=\text{tokenptr}\]

The name (RS-type), or address in register (2)-(12), of a required 4-byte input that contains the address of a 8-byte area that contains a token uniquely identifying this instance of a BLOCK request.

**UNBLOCK**

Resume Basic HyperSwap activities.

\[\text{NAMEPTR}=\text{nameptr}\]

The name (RS-type), or address in register (2)-(12), of a required 4-byte input that contains the address of a 8-character area that contains the name of the application issuing the UNBLOCK request. If this name does not match the name of the application for which Basic HyperSwap is currently blocked, the UNBLOCK request will be rejected.

\[\text{TOKENPTR}=\text{tokenptr}\]

The name (RS-type), or address in register (2)-(12), of a required 4-byte input that contains the address of a 8-byte area that contains the token specified on the previous BLOCK request. If this token does not match the token specified on the BLOCK request for which Basic HyperSwap is currently blocked or if FORCE=YES is not specified on this UNBLOCK request, the request will be rejected.
IOSHXBLK macro

,FORCE=YES

An optional keyword input that specifies whether to allow unblock if the
token value on an UNBLOCK request does not match the token value on
the BLOCK request for which Basic HyperSwap is currently blocked. The
default is NO.

,FORCE=NO

Even though the token specified on this request does not match the
token specified on the BLOCK request for which Basic HyperSwap is
currently blocked, honor this UNBLOCK request as long as the name
specified on this request matches the name specified on the BLOCK
request for which Basic HyperSwap is currently blocked.

,FORCE=NO

Do not unblock unless name and token specified on this UNBLOCK
request match the name and token specified on the BLOCK request for
which Basic HyperSwap is currently blocked.

TESTBLOCK

Test whether all systems in the SYSPLEX support programmatic blocking.

End of mutually exclusive required keys.

,RETCODE=retcode

The name (RS-type) of an optional full-word output variable, or register
(2)-(12) or (15), into which the return code is to be copied from GPR 15. If you
specify 15, GPR15, REG15, or R15 (within or without parentheses), the value
will be left in GPR 15.

,RSNCODE=rsncode

The name (RS-type) of an optional full-word output variable, or register
(2)-(12), into which the reason code is to be copied from GPR 0. If you specify
0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the
value will be left in GPR 0.

,PLISTVER=plistver

An optional byte input decimal value in the "1-1" range that specifies the
macro version. PLISTVER is the only key allowed on the list form of MF and
determines which parameter list is generated. Note that MAX may be specified
instead of a number, and the parameter list will be of the largest size currently
supported. This size may grow from release to release (thus possibly affecting
the amount of storage needed by your program). If your program can tolerate
this, IBM recommends that you always specify MAX when creating the list
form parameter list as that will ensure that the list form parameter list is
always long enough to hold whatever parameters might be specified on the
execute form.

The default is IMPLIED_VERSION. When PLISTVER is omitted, the default is
the lowest version which allows all of the parameters specified on the
invocation to be processed.

,MF=S

An optional keyword input which specifies the macro form. The default is S.

,MF=(L,mfctrl, mfattr | 0D)

,MF=(E,mfctrl,COMPLETE)

Specifies the standard form of the macro. The 'S' form generates code to
put the parameters into an in-line parameter list and invoke the desired service. Full checking for required macro keys is done along with supplying defaults for omitted optional parameters.

```
,MF=(L,mfctrl,0D)
```

Specifies the list form of the macro. The 'L' form defines an area to be used for the parameter list. Only the PLISTVER key may be specified on the invocation. All other macro parameters are flagged as errors. If PLISTVER is not specified, the original parameter list definition is used.

```
,MF=(E,mfctrl,COMPLETE)
```

Specifies the execute form of the macro. The 'E' form generates code to put the parameters into the parameter list specified by mfctrl and provides full syntax checking with default setting.

```
,MF=(L,mfctrl,0D)
```

A required input. It is the name of a storage area for the parameter list.

```
,MF=(E,mfctrl,COMPLETE)
```

An optional 60-character input string that varies from 1 to 60 characters. Use it to force boundary alignment of the parameter list. Use only 0F or 0D. The default is 0D, which forces the parameter list to a doubleword boundary.

```
,MF=(L,mfctrl,0D)
```

A required input. It is the name (RS-type), or address in register (1)-(12), of a storage area for the parameter list.

```
,MF=(E,mfctrl,COMPLETE)
```

An optional keyword input which specifies the degree of macro parameter syntax checking. The default is COMPLETE.

Checking for required macro keys is done and defaults are supplied for omitted optional parameters.

### ABEND codes

None.

### Return and reason codes

When the IOSHXBLK macro returns control to your program:

- GPR 15 (and retcode, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes:

<table>
<thead>
<tr>
<th>Hexadecimal return code</th>
<th>Reason codes, meaning and action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Basic HyperSwap services have successfully completed the requested action. For TESTBLOCK requests, all systems support programmatic blocking.</td>
</tr>
</tbody>
</table>
## IOSHXBLK Macro

### Table 89. Return Codes for the IOSHXBLK Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal return code</th>
<th>Reason codes, meaning and action</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>Basic HyperSwap services did not complete the requested action. For TESTBLOCK requests, at least one system does not support programmatic blocking.</td>
</tr>
<tr>
<td></td>
<td><strong>Reason code</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning/Action</strong></td>
</tr>
<tr>
<td>00</td>
<td>Basic HyperSwap services not started.</td>
</tr>
<tr>
<td>01</td>
<td>Basic HyperSwap services initialization incomplete.</td>
</tr>
<tr>
<td>41</td>
<td>HyperSwap in progress.</td>
</tr>
<tr>
<td>42</td>
<td>Basic HyperSwap services are temporarily busy.</td>
</tr>
<tr>
<td>43</td>
<td>Basic HyperSwap blocking services are in unknown state due to a system error. Unblock with Force option is required to clear this condition.</td>
</tr>
<tr>
<td>44</td>
<td>IOSHXBLK service timed out waiting for Basic HyperSwap blocking services to complete the request. Unblock with Force option may be required before retrying the request.</td>
</tr>
<tr>
<td>08</td>
<td>Basic HyperSwap services did not accept the request.</td>
</tr>
<tr>
<td></td>
<td><strong>Reason code</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning/Action</strong></td>
</tr>
<tr>
<td>01</td>
<td>Invalid function code.</td>
</tr>
<tr>
<td>02</td>
<td>Caller is not in task mode.</td>
</tr>
<tr>
<td>03</td>
<td>Caller is not in supervisor state.</td>
</tr>
<tr>
<td>04</td>
<td>Caller is not in primary ASC mode.</td>
</tr>
<tr>
<td>05</td>
<td>Caller is not enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>06</td>
<td>Caller is holding lock(s).</td>
</tr>
<tr>
<td>07</td>
<td>System resources unavailable to copy request.</td>
</tr>
<tr>
<td>08</td>
<td>Internal function code mismatch.</td>
</tr>
<tr>
<td>09</td>
<td>Routing of request in SYSPLEX failed.</td>
</tr>
<tr>
<td>0A</td>
<td>System service error occurred while processing the request.</td>
</tr>
<tr>
<td>0B</td>
<td>Blocking not supported by Basic HyperSwap master system.</td>
</tr>
<tr>
<td>0C</td>
<td>Blocking not supported by one or more Basic HyperSwap member systems.</td>
</tr>
<tr>
<td>0D</td>
<td>HyperSwap API services address space is not active on one or more Basic HyperSwap member systems.</td>
</tr>
</tbody>
</table>
### Table 89. Return Codes for the IOSHXBLK Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal return code</th>
<th>Reason codes, meaning and action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C</td>
<td>Programming error.</td>
</tr>
<tr>
<td></td>
<td><strong>Reason code</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning/Action</strong></td>
</tr>
<tr>
<td>01</td>
<td>BLOCK request while already blocked. The name pointed to by NAMEPTR on the BLOCK request is the same as the name of the application currently blocking Basic HyperSwap.</td>
</tr>
<tr>
<td>02</td>
<td>BLOCK request while already blocked. The name pointed to by NAMEPTR on the BLOCK request is different from the name of the application currently blocking Basic HyperSwap.</td>
</tr>
<tr>
<td>03</td>
<td>UNBLOCK request while not blocked.</td>
</tr>
<tr>
<td>04</td>
<td>Name or token specified on an UNBLOCK request did not match name or token specified on the BLOCK request that is currently blocking Basic HyperSwap.</td>
</tr>
<tr>
<td>05</td>
<td>Not a BLOCK, UNBLOCK or TESTBLOCK request.</td>
</tr>
<tr>
<td>10</td>
<td>Environmental error.</td>
</tr>
<tr>
<td></td>
<td><strong>Reason code</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Meaning/Action</strong></td>
</tr>
<tr>
<td>00</td>
<td>Caller not authorized or invocation environment wrong.</td>
</tr>
<tr>
<td>01</td>
<td>Unable to establish a recovery environment.</td>
</tr>
<tr>
<td>02</td>
<td>System resources unavailable for the request.</td>
</tr>
<tr>
<td>03</td>
<td>ALESERV service failed for the request.</td>
</tr>
<tr>
<td>20</td>
<td>System abend occurred while processing request.</td>
</tr>
</tbody>
</table>
IOSHXBLK macro
Chapter 99. IOSINFO — Obtain the subchannel number for a UCB

Description

The IOSINFO macro obtains the subchannel number for a specified unit control block (UCB). The macro returns the subsystem identification word (SID), which identifies the subchannel number of the UCB, in a user-specified location. The SID is a fullword value whose first halfword contains X’0001’ and ending halfword contains the subchannel number.

Environment

The issuer of IOSINFO must be executing:
• In 31-bit addressing mode
• In either task mode or SRB mode
• Locked or unlocked

Additionally, the issuing program must include the CVT and IHAPSA mapping macros. All addresses must be 31-bit addresses and the issuing program must pass a below 16 megabyte actual or captured UCB.

Input register information

Before entry to this macro, register 13 must contain the address of a standard 18-word save area.

Output register information

After the caller issues the macro, the macro might use some registers as work registers or might change the contents of some registers. When the macro returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the macro</td>
</tr>
<tr>
<td>1</td>
<td>Contains the SID if the return code in register 15 is 0; otherwise, used as a work register by the macro.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the macro</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>
IOSINFO macro

Syntax
The IOSINFO macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSINFO.</td>
</tr>
<tr>
<td>IOSINFO</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSINFO.</td>
</tr>
<tr>
<td>FUNCTN=SUBCHNO</td>
<td></td>
</tr>
<tr>
<td>,UCB=ucb addr</td>
<td>ucb addr: A-type address or register (0) - (15).</td>
</tr>
<tr>
<td>,OUTPUT=output addr</td>
<td>output addr: A-type address or register (0) - (14).</td>
</tr>
<tr>
<td>,RTNCODE=retcde addr</td>
<td>retcde addr: A-type address or register (0) - (15).</td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained as follows:

- **FUNCTN=SUBCHNO**
  Specifies that a subchannel number is to be obtained.

- **,UCB=ucb addr**
  Specifies the address of a fullword on a fullword boundary containing the address of a unit control block (UCB).

- **,OUTPUT=output addr**
  Specifies the address of a fullword on a fullword boundary that will contain the subsystem identification word (SID) upon completion.
  The SID is a fullword value that identifies the subchannel. The first halfword is X’0001’, and the last halfword contains the subchannel number.
  The output address must reside in 31-bit addressable storage.

- **,RTNCODE=retcde addr**
  Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15. The specified storage location must be a fullword on a fullword boundary.
  The return code address must reside in 31-bit addressable storage.

Return codes
When control returns from IOSINFO, GPR 15 (and retcde addr, if you coded RTNCODE) contains one of the following return codes:
**Hexadecimal Code**

<table>
<thead>
<tr>
<th>Hexadecimal Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>The address specified on the OUTPUT parameter contains the SID.*</td>
</tr>
<tr>
<td>04</td>
<td>The UCB was disassociated from the subchannel at the time of the IOSINFO service routine invocation.</td>
</tr>
</tbody>
</table>

* In some cases, the subchannel number in the SID might not be valid. Any disassociation of the UCB and the subchannel means the subchannel number in the SID is not valid. If the UCB is disassociated from the subchannel after the IOSINFO service routine invocation, no notification can be given.

**Example 1**

Obtain the subchannel number for a UCB whose address is in register 1. Specify the SID output to be placed in register 2 and the return code to be placed in register 3.

```ml
IOSINFO FUNCTN=SUBCHNO,UCB=(1),OUTPUT=(2),RTNCODE=(3)
```

**Example 2**

Obtain the subchannel number for a UCB whose address is in location ADDR. Specify the SID output to be placed in location ADDX and the return code to be placed in register 3.

```ml
IOSINFO FUNCTN=SUBCHNO,UCB=ADDR,OUTPUT=ADDX,RTNCODE=(3)
```

**Example 3**

Obtain the subchannel number for a UCB whose address is in register 2. Specify the SID output to be placed in register 3 and the return code to be placed in location ADDR.

```ml
IOSINFO FUNCTN=SUBCHNO,UCB=(2),OUTPUT=(3),RTNCODE=ADDR
```
IOSINFO macro
Chapter 100. IOSLOOK — Locate unit control block

Note: The UCBLOOK macro is the preferred programming interface.

Description

The IOSLOOK macro locates the unit control block (UCB) associated with a device number. To use IOSLOOK, you must be executing in supervisor state. Register 13 must point to a 16-word save area where the macro stores registers 0 through 15 at offset 0. You must also include a DSECT for both the CVT (using the CVT mapping macro) and the IOCOM (using the IECDIOCM mapping macro).

Syntax

The IOSLOOK macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>DEV=(reg)</td>
<td>reg: Register (0) - (12), (14), (15). Default: DEV=(6).</td>
</tr>
<tr>
<td>,UCB=(reg)</td>
<td>reg: Register (0) - (12). Default: UCB=(7).</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

DEV=(reg)

Specifies a general purpose register, symbolic or absolute, that contains the hexadecimal device number, right justified. If this parameter is omitted, register 6 is assumed.

,UCB=(reg)

Specifies a general purpose register, symbolic or absolute, that will be used to return the address of the UCB common segment. If this parameter is omitted, register 7 is assumed. If the UCB address cannot be found, then the contents of this register are unpredictable.

Note: The UCB must reside in 24-bit addressable storage.
Return codes

When IOSLOOK macro returns control to your program, GPR 15 contains a return code.

Table 90. Return Codes for the IOSLOOK Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: UCB address was found.</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: Device number was invalid or no UCB exists.</td>
</tr>
</tbody>
</table>

Example

Find the UCB address for device 250. Register 2 contains the value X'00000250'. The UCB address is to be returned in register 5 and UCBPTR is equated to 5.

IOSLOOK DEV=(2),UCB=(UCBPTR)
Chapter 101. IOSODS — IOS offline device service

Description

The IOS Offline Device Service macro provides the interface for authorized code to mark a device offline and in use by a system component.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state and key 0.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task mode.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>None</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

- The caller may not hold any locks.
- The caller is required to pin the UCB for the device before invoking the IOSODS macro service. Pinning the UCB will insure that the proper identification of the user of the device will be displayed if the installation should try the dynamically delete it.
- Issuers of the IOSODS macro service must provide recovery and resource termination managers to insure that the device is freed for use by other applications in case of an unexpected failure or cancellation of the address space.

Input register information

None.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1–14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:
IOSODS macro

Register

Contents

0-15 Unchanged

Performance implications
None.

Syntax
The standard form of the IOSODS macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>△</td>
<td>One or more blanks must precede IOSODS.</td>
</tr>
<tr>
<td>IOSODS</td>
<td></td>
</tr>
<tr>
<td>△</td>
<td>One or more blanks must follow IOSODS.</td>
</tr>
<tr>
<td>OFF</td>
<td>Default: None.</td>
</tr>
<tr>
<td>[,WLMPAVSUSPEND]</td>
<td>This function is disabled.</td>
</tr>
<tr>
<td>[,WLMPAVRESTORE]</td>
<td>This function is disabled.</td>
</tr>
<tr>
<td>,DEVN=devn</td>
<td>devn: RS-type name or address in register (2) - (12).</td>
</tr>
<tr>
<td>[,DEVNCHAR=devnchar]</td>
<td>devnchar: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[SCHSET=schset</td>
<td>0]</td>
</tr>
<tr>
<td>[,LDEVNCHAR=ldevnchar]</td>
<td>ldevnchar: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[,RETCODE=retcode]</td>
<td>retcode: RS-type name or register (2) - (12).</td>
</tr>
<tr>
<td>[,RSNCODE=rsncode]</td>
<td>rsncode: RS-type name or register (2) - (12).</td>
</tr>
<tr>
<td>[,PLISTVER={plistver</td>
<td>MAX</td>
</tr>
</tbody>
</table>
Parameters

The parameters are explained as follows:

The following is a set of mutually exclusive keywords. This set is required; only one keyword must be specified.

**ON**
Keyword that indicates the input device number is to be marked as offline and in use by a system component.

**WLMPAVSUSPEND**
Optional keyword that indicates that the Work Load Manager dynamic alias tuning capability for the device (if applicable) will be suspended. Note that it is up to the user to restore this capability through an IOSODS OFF request with the WLMPavRestore keyword specified. This function is currently disabled.

**Default:** None.

**OFF**
Keyword that indicates the input device number is no longer to be marked as offline and in use by a system component.

**WLMPAVRESTORE**
Optional keyword that indicates that the Work Load Manager dynamic alias tuning capability for the device (if applicable) will be restored. Note that it is up to the user to restore this capability through an IOSODS ON request with the WLMPavSuspend keyword specified. This function is currently disabled.

**Default:** None.

This ends the set of mutually exclusive required keywords.

**DEVN=devn**
The name (RS-type), or address in register (2) - (12), of a required halfword input that specifies the device number in binary of the device that is to be operated on.

**DEVNCHAR=devnchar**
The name (RS-type), or address in register (2) - (12), of an optional byte input that specifies the device number in EBCDIC, of the device that is to be operated on.

**SCHSET=schset | 0**
The name (RS-type), or address in register (2) - (12), of an optional byte input that specifies the subchannel set of the device. The default is 0.

**LDEVNCHAR=ldevnchar**
The name (RS-type), or address in register (2) - (12), of a 5-character input that specifies the logical device number, in EBCDIC, of the device whose UCB address is to be obtained.

**Note:** A logical device number is represented by a 1-digit subchannel set id followed by the 4-digit device number, sdddd.

**RETCODE=retcode**
The name (RS-type), or register (2) - (12), of an optional fullword output into which the return code is to be copied from GPR 15.
**IOSODS macro**

,**RSNCODE=rsncode**
The name (RS-type), or register (2) - (12), of an optional fullword output into which the reason code is to be copied from GPR 0.

,**PLISTVER=plistver | MAX | IMPLIED_VERSION**
is an optional byte input decimal value in the “1–1” range that specifies the macro version. PLISTVER is the only key allowed on the list form of MF and determines which parameter list is generated. Note that MAX may be specified instead of a number and will cause the parameter list to be of the largest size currently supported. This size may grow from release to release (thus possibly affecting the amount of storage needed by your program). If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as this will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

**Default:** IMPLIED_VERSION. When PLISTVER is omitted, the default is the lowest version which allows all of the parameters specified on the invocation to be processed.

**ABEND codes**
None.

**Return codes**
Return and reason codes, in hexadecimal, from the IOSODS macro are as follows:

<table>
<thead>
<tr>
<th>Hexadecimal return code</th>
<th>Hexadecimal reason code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None</td>
<td>The requested function executed successfully.</td>
</tr>
<tr>
<td>08</td>
<td>01</td>
<td>The requested function failed because the input device number was not found.</td>
</tr>
<tr>
<td>08</td>
<td>02</td>
<td>A request to allocate an offline device was made, but it is already in use by a system component (ON function).</td>
</tr>
<tr>
<td>08</td>
<td>03</td>
<td>A request to unallocate a device was made, but the device is not currently in use (OFF function).</td>
</tr>
<tr>
<td>08</td>
<td>04</td>
<td>IOS Path Validation failed (ON function).</td>
</tr>
<tr>
<td>08</td>
<td>05</td>
<td>IOS Dynamic Pathing function failed (OFF function).</td>
</tr>
<tr>
<td>08</td>
<td>06</td>
<td>Allocation service to set UCBNALOC failed (ON function).</td>
</tr>
<tr>
<td>20</td>
<td>None</td>
<td>An unexpected error occurred. The recovery routine recovered and returned control to the caller.</td>
</tr>
</tbody>
</table>

**IOSODS - List form**

Use the list form of the IOSODS macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses to contain the parameters.
IOSODS macro

Syntax

The list form of the IOSODS macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin \textit{name} in column 1.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>[</td>
<td>One or more blanks must precede IOSODS.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>IOSODS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>One or more blanks must follow IOSODS.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>[.\text{PLISTVER}={xplistver \mid \text{MAX} \mid \text{IMPLIED_VERSION}}]</td>
<td>\textbf{Default: IMPLIED_VERSION}</td>
</tr>
<tr>
<td>[.\text{MF}=(L,\text{xmfctrl},{,\text{xmfattr}</td>
<td>0D})]</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained under the standard form of the IOSODS macro, with the following exception:

\texttt{,MF=(L,\text{xmfctrl},\{,\text{xmfattr}|0D\})}

\textit{L} specifies the list form of the macro. The list form defines an area to be used for the parameter list. Only the PLISTVER key may be specified on the invocation. All other macro parameters are flagged as errors. If PLISTVER is not specified, the original parameter list definition is used.

\texttt{,\text{xmfctrl}}

This required input is the RS-type name, or address in register (1) - (12), of a storage area for the parameter list.

\texttt{,\text{xmfattr}|0D}

This is an optional 60 character input string which is used to force boundary alignment of the parameter list. Use only \texttt{0F} or \texttt{0D}.

\textbf{Default: 0D} which forces the parameter list to a doubleword boundary.

IOSODS - Execute form

Use the execute form of the IOSODS macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.
The execute form of the IOSODS macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td></td>
<td>One or more blanks must precede IOSODS.</td>
</tr>
<tr>
<td>IOSODS</td>
<td>One or more blanks must follow IOSODS.</td>
</tr>
<tr>
<td>ON</td>
<td>Default: None.</td>
</tr>
<tr>
<td>[,WLMPAVSUSPEND]</td>
<td>This function is disabled.</td>
</tr>
<tr>
<td>OFF</td>
<td>Default: None.</td>
</tr>
<tr>
<td>[,WLMPAVRESTORE]</td>
<td>This function is disabled.</td>
</tr>
<tr>
<td>,DEVN=devn</td>
<td>devn: RS-type name or address in register (2) - (12).</td>
</tr>
<tr>
<td>[,RETCODE=retcode]</td>
<td>retcode: RS-type name or register (2) - (12).</td>
</tr>
<tr>
<td>[,RSNCODE=rsncode]</td>
<td>rsncode: RS-type name or register (2) - (12).</td>
</tr>
<tr>
<td>[,PLISTVER={xplistver</td>
<td>MAX</td>
</tr>
<tr>
<td>[,MF=(E,xmfctrl[,COMPLETE])]</td>
<td>xmfctrl: RS-type name or address in register (2) - (12). Default: COMPLETE</td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained under the standard form of the IOSODS macro, with the following exception:

,MF=(E,xmfctrl[,COMPLETE])
E specifies the execute form of the macro. The execute form generates code to put the parameters into the parameter list specified by xmfctrl and provides full syntax checking with default setting.

,xmfctrl
This required input is the RS-type name, or address in register (1) - (12), of a storage area for the parameter list.
IOSODS macro

,COMPLETE
   An optional keyword which specifies the degree of macro parameter syntax checking. When complete checking is enabled, required parameters are checked and defaults are supplied for omitted optional parameters.

Default: COMPLETE
Chapter 102. IOSPTHV — Validate I/O paths

Description

The IOSPTHV macro enables authorized callers to validate the physical connectivity and availability of a channel path to a device. A path is considered available if an I/O operation can be initiated down a path, and the device can be selected. Validation does not guarantee that the device and path are error free. The IOSPTHV function depends on the availability of the IOS Address Space (IOSAS). IOSAS is started after Master Scheduler Initialization (MSI), and may be unavailable for periods of time during recovery. The issuer of the IOSPTHV macro must be able to handle the return/reason code indicating that the IOSAS is not active.

IOSPTHV is similar to UCBINFO PATHINFO and the VARY command, but there are important differences. UCBINFO returns status based on UCB indicators that might be outdated. Unlike the VARY command, IOSPTHV does not change UCB path status indicators or dynamically vary paths online or offline. IOSPTHV only tests physical connectivity. Examples of when you might validate a path include:

- To verify the current availability of a specific path to a specific device and present any path-related errors to a user.
- As a first step in diagnosing path-related problems.
- After installing a device, to verify channel to device connectivity before issuing a VARY command for the device.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Authorization</td>
<td>Supervisor state and any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be in the primary address space</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

Do not have any enabled, unlocked task (EUT) FRRs established. If issued during IPL before the IOSAS (IOS address space) has initialized, MSI must have completed and WAIT=YES must be specified on the IOSPTHV macro.

If you attempt to validate a path to an active teleprocessing device (device types 2701, 2702, and 2703) or to an OSA or CTC device in use by VTAM with a long running I/O active, you will receive an error return and reason code.
Input register information
Before issuing the IOSPTHV macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller of the IOSPTHV macro, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after invoking a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
The standard form of the IOSPTHV macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSPTHV.</td>
</tr>
<tr>
<td>IOSPTHV</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSPTHV.</td>
</tr>
<tr>
<td>,DEVN=</td>
<td></td>
</tr>
<tr>
<td>device number</td>
<td>device number: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>,CHPID=path id</td>
<td>path id: RX-type address or address in register (2) - (12).</td>
</tr>
</tbody>
</table>
### Parameter descriptions

The parameters are explained as follows:

**,DEVN=** *device number*

Specifies the device's binary device number (0000 - FFFF). IOSPTHV checks the availability of the path you specify on the CHPID parameter to the device you specify on DEVN. IOSPTHV pins the device so that the device's UCB and other related data structures are not dynamically deleted while IOSPTHV is validating the path. When IOSPTHV completes processing, it unpins the device.

**,CHPID=** *path id*

Specifies the ID (00 - FF) of the channel path that IOSPTHV validates for physical availability. To determine the ID for a specific channel path connected to the device specified on DEVN, use the UCBINFO PATHINFO macro or the DISPLAY MATRIX operator command.

**,MSGBUF=** *msgbuf addr*

Specifies the address of a 71-character area into which IOSPTHV is to place diagnosis information. IOSPTHV uses this buffer only if the return code is 'X'04' and the reason code is 'X'04'. This information consists of the same message that is issued by the VARY command for comparable errors. This message is the last message that MVS would have issued if a VARY PATH command had been issued and a similar error had been encountered. IOSPTHV does not issue a message to the operator console.

**,IOCTOKEN=** *ioctoken addr*

Specifies the address of a 48-character area that contains the MVS I/O configuration token that you supply to IOSPTHV. You can obtain this token by issuing the IOCINFO macro, which is described in *[z/OS MVS Programming - Assembler Services Reference ABE-HSP]*. If the I/O configuration token that is current when IOSPTHV is invoked does not match the token you supply, you are notified through a return code.
If you set the input IOCTOKEN to binary zeros, IOSPTHV sets IOCTOKEN to the current I/O configuration token.

For information about how you can use the configuration token to detect configuration changes, see z/OS MVS Programming: Authorized Assembler Services Guide.

\[\text{,WAIT=NO}\]
\[\text{,WAIT=YES}\]

Is an optional keyword input that indicates to allow the request to wait for the IOS address space (IOSAS) to initialize or restart (if terminated) before continuing. WAIT=NO is the default.

WAIT=NO: Only process if the IOS address space has been initialized and not terminated.

WAIT=YES: Allows the request to wait for the IOSAS space to initialize as long as MSI has completed. Allows the request to wait for the IOS address space to reinitialize if terminated. The user of this keyword must ensure that no no resources are held that can cause the IOSAS not to initialize.

\[\text{,TIME=\text{time}}\]

Specifies an 8-byte field containing the maximum amount of time, in seconds, that IOSPTHV can run before being purged. The default for the TIME parameter is 5 seconds.

The time interval, whose address resides in virtual storage, is presented as zoned decimal digits of the form:

\[\text{HHMMSS}t\text{h}\]

where:

- **HH** is hours (24-hour clock)
- **MM** is minutes
- **SS** is seconds
- **t** is tenths of seconds
- **h** is hundredths of seconds

IOSPTHV runs until one of the following occurs:

- IOSPTHV completes successfully or unsuccessfully
- The interval that you specify on the TIME= parameter expires
- The MIH interval for the device expires.

Note that the TIME parameter allows you to set an expiration time that is specific to IOSPTHV. The MIH interval, however, is used by other services associated with the device. Using the TIME parameter allows you to set an expiration time that is shorter than the MIH interval or the time that it takes the IOSPTHV macro to complete successfully.

\[\text{,RETCODE=rc}\]

Specifies the location or register where the system is to place the return code. The system copies the return code into the location from register 15.

\[\text{,RSNCODE=rsncode}\]

Specifies the location or register where the system is to place the reason code. The system copies the reason code into the location from register 0.
## Return and reason codes

Return codes, in hexadecimal, from the IOSPTHV macro are as follows:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | None                    | Meaning: IOSPTHV processing completed successfully. IOSPTHV successfully validated the specified path. The path is physically available.  
Action: None |
| 04 04                   | 04                      | Meaning: IOSPTHV did not successfully validate the specified path because the path was not physically available.  
Action: You need to investigate the problem further. Trying to vary the path online may produce further diagnosis data. |
| 04 08                   | 08                      | Meaning: User-specified time interval on the TIME keyword expired before the I/O completed.  
Action: Verify that the time interval was long enough. Note that this return code is issued only if the time expired before the MIH interval. You can use the D MIH command or the MIQUERY macro to determine the MIH interval for the device. |
| 08 04                   | 04                      | Meaning: IOSPTHV did not successfully validate the specified path because the device number specified on the DEVN keyword is not valid.  
Action: Ensure that you specified the device number correctly and retry the operation. Use the IOCTOKEN keyword to ensure that the UCB for that device number was not dynamically changed or deleted. |
| 08 08                   | 08                      | Meaning: IOSPTHV did not successfully validate the specified path because the path specified on the CHPID keyword is not valid.  
Action: Verify your program to ensure that the correct CHPID was passed and retry the operation. Use the IOCTOKEN keyword to ensure that the CHPID for the device was not dynamically changed or deleted. |
| 08 0C                   | 0C                      | Meaning: IOSPTHV did not successfully validate the specified path because the time specified on the TIME keyword was not valid.  
Action: Ensure that the time specified contains valid zoned decimal digits in the proper range. |
| 08 20                   | 20                      | Meaning: IOSPTHV did not successfully validate the specified path because the UCB definition for the device represented by the look-up argument (device number) has changed and is no longer consistent with the UCB definition represented by the input I/O configuration token. (This return code is only valid for callers using the IOCTOKEN keyword.)  
Action: Ensure that the device number and CHPID are still valid and retry the operation passing a current IOCTOKEN. |
**Example**

Determine if a channel path to the SYSRES device is available without changing the online/offline status of the path. Scan through all UCBs, using the UCBSCAN macro, and put copies of the DASD UCBs the program finds in a user-supplied work area called UCBSTOR. When the program finds the SYSRES device, issue the UCBINFO macro to obtain information about the device path and type of channel path for the specified UCB. Information, such as the channel path ID and online status, will appear in the IOSDPATH data area. The program looks through the channel path information until it finds an online path. Issue the IOSPTHV macro to test whether the online path is available.

```
IOSPTHV CSECT
IOSPTHV AMODE 31 31-BIT ADDRESSING MODE
IOSPTHV RMODE ANY Rmode any
SPACE 1

*.....................................................................*
* REGISTER ASSIGNMENTS                                      *
*.....................................................................*
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6     Dynamic area register
UCBPTR7 EQU 7   UCB Pointer
R8 EQU 8
R9 EQU 9     Module base register
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13     Pointer to standard save area
R14 EQU 14
R15 EQU 15
SPACE 3
TITLE 'IOSPTHV - IOSPTHV Sample Program'

*.....................................................................*
* Standard Entry Linkage                                    *
*.....................................................................*
PRINT GEN
```
**IOSPTHV macro**

- **USING *,R9**  
  Sets up base register

- **ENTRY STM R14,R12,12(R13)**  
  Save caller's registers

- **LR R9,R15**  
  Establish module base register

- **MODESET KEY=ZERO,MODE=SUP**

- **LA R0,DYNSIZE**  
  Load length of dynamic area

- **STORAGE OBTAIN,LENGTH=((R0)),SP=233**  
  Gets dynamic area

- **LR R6,R1**  
  Gets dynamic area address

- **USING DYNAREA,R6**  
  Sets up dynamic area

- **ST R13,SAVE+4**  
  Save caller's save area address

- **LA R15,SAVE**  
  Get this module's save area address

- **ST R15,8(R13)**  
  Save this module's save area address in caller's save area.

- **LR R13,R15**  
  Set up addressability to this module's save area.

- **B MAINLINE**

- **DC CL8'IOSSPTHV'**

- **DC CL8'SYSDATE'**

- **DC CL8'SYSTIME'**

- **TITLE 'IOSSPTHV - SPTHV mainline'**

- **L 10,X'10'**  
  Load CVT pointer

- **USING CVT,10**

- **TM CVTDCB,CVTOSEXT**  
  Is the OSLEVEL extension present

- **BNO NO_IOSPTHV**  
  No, pre-MVS/SP Version 3 system

- **TM CVTOSLV1,CVTH5510**  
  Running on version HBB5510?

- **BNO NO_IOSPTHV**  
  No, pre-HBB5510 system. IOSPTHV supported on HBB5510 and above

- **LA UCBPTR7,UCBSTOR**  
  Get address of work area

- **USING UCB,UCBPTR7**  
  Set up addressability

- **LA R0,SCANWORK**  
  Set storage address

- **LA R1,100**  
  Set storage length

- **SR R15,R15**  
  Clear second operand

- **MVCL R0,R14**  
  Clear the storage

- **SCANLOOP UCBSCAN COPY, X**
  
  WORKAREA=SCANWORK, X
  
  UCBAREA=UCBSTOR, X
  
  DEVCCLASS=DASD, X
  
  MF=(E,SCANLIST)
If UCBSCAN returned a UCB, check whether the UCB represents the SYSRES volume. If it isn't, continue checking more UCBs. If the UCB represents the SYSRES device, end the loop.

LTR R15,R15 Test return code
BNZ EXIT_ERROR Exit if non-zero
TM UCBSTAT,UCBSYSR Test if SYSRES volume
BZ SCANLOOP Keep looping if not

Issue the UCBINFO macro to obtain path-related information. UCBINFO returns this information in a field called PATHTOR, mapped by IOSDPATH.

Note- Since the device whose path information is sought is the SYSRES device, an online path is certain to be found. No loop counter is used.

UCBINFO PATHINFO, X
DEVN=UCBCHAN, X
PATHAREA=PATHSTOR, X
MF=(E,INFOLIST)

If UCBINFO cannot retrieve path-related information, that is, you receive a non-zero return code, exit program.

LTR R15,R15 Test for 0 return code
BNZ EXIT_ERROR Exit if bad RC

Loop through the channel path ID array entries returned in PATHSTOR to find the first online path. An online path is represented by a flag in the array.

LA R10,PATHSTOR Address of PATHINFO data
USING PATH,R10 Set up addressability to path information.
SR R8,R8 CHPID array index register.

CHPID_LOOP IC R11,PATHBITS(R8) Get flags from array entry.
STC R11,PATHSAVE Save entry
TM PATHSAVE,X'04' Test if the path is online
BO CHPID_EXIT If so, exit the loop
LA R8,L'PATHCHPIDARRAY(R8) Increment array index
B CHPID_LOOP

CHPID_EXIT LH R11,PATHCHPID(R8) Get the ID for the online channel path.
STC R11,CHPID Save the ID for the online channel path.

Test the availability of the first online path to the SYSRES volume by issuing the IOSPTHV macro. Supply the channel path ID of the online path on the CHPID parameter.

Note: Although the logical path mask (LPM) indicated that the path was logically online to the device, it is possible that the path is not operational. IOSPTHV performs an I/O operation down the path to determine if a non-operational condition exits.
IOSPTHV macro

*.....................................................................* 
IOSPTHV DEVN=UCBCCHAN,  
CHPID=CHPID,  
MF=(E,PTHVLIST)  
*.....................................................................*

* A zero return code indicates an operational path to  
* the specified device. A non-zero return code indicates  
* a non-operational path. In the latter case, examine the  
* return and reason code to determine the cause.  
*                                               
*.....................................................................* 
LTR R15,R15 
BZ PATH_OK 
B PATH_NOK  
PATH_OK  
DS 0D 
WTO 'IOSPTHV-FIRST ONLINE PATH TO SYSRES VALIDATED',  
ROUTCDE=(11),DESC=(2)  
B EXIT 
PATH_NOK  
DS 0D 
WTO 'IOSPTHV-FIRST ONLINE PATH TO SYSRES NOT VALIDATED',  
ROUTCDE=(11),DESC=(2)  
B EXIT  
*.....................................................................* 
NO_IOSPTHV  
DS 0H 
WTO 'IOSPTHV - IOSPTHV SUPPORTED IN HBB5510 AND HIGHER',  
ROUTCDE=(11),DESC=(2)  
B EXIT  
*.....................................................................* 
EXIT_ERROR  
DS 0H 
WTO 'IOSPTHV - THE SAMPLE ENCOUNTERED AN ERROR',  
ROUTCDE=(11),DESC=(2)  
*.....................................................................* 
EXIT  
DS 0H 
L R13,SAVE+4  
LA R0,DYNSIZE  
STORAGE RELEASE,SP=233,ADDR=(R6),LENGTH=(R0)  
MODESET KEY=NZERO,MODE=PROB 
LM R14,R12,I2(R13)  
BR R14  
*.....................................................................* 

Chapter 102. IOSPTHV — Validate I/O paths 1047
**IOSPTHV macro**

```assembly
*......................................................................*
DYNSTART   DS 0H
DYNAREA    DSECT
  * Save area
SAVE       DS 18F
  DS 0D       Force doubleword alignment
SPACE 2

*......................................................................*
* List forms of macros. The list and execute forms of these macros *
* are used because this module is reentrant. *
*......................................................................*
LIST_INFOSERV UCBINFO MF=(L,INFOLIST) List form of UCBINFO
INFOSERV_END DS 0D
PATHSTOR   DS CL256 Storage for the PATHAREA
PATHSTOR_END DS 0D
LIST_PTHVSERV IOSPTHV MF=(L,PTHVLIST) List form of IOSPTHV
PTHVSERV_END DS 0D
LIST_SCANSERV UCBSCAN MF=(L,SCANLIST) List form of UCBSCAN
SCANSERV_END DS 0D
SCANWORK   DS CL100 Scan work area
SCANWORK_END DS 0D
UCBSTOR    DS CL48 UCB copy storage
UCBSTOR_END DS 0D

*......................................................................*
* Work variables and data structures local to this module *
*......................................................................*
CHPID      DS C CHPID used for IOSCDR invocation
PATHSAVE   DS C Work variable for CHPID array
  * entries in the PATHAREA.
END_DYN    DS 0D
DYN_SIZE   EQU *-DYNAREA Calculates Dynamic area

*......................................................................*
* DSECTs *
*......................................................................*
IOSSPATHV  CSECT
  TITLE 'IOSSPATHV - DSECT MAPPINGS'
EJECT
CVT LIST=YES,DSECT=YES

UCB DSECT
  IEFUCBOB

PATHAREA IOSPATH
END IOSSPATHV
```
Chapter 103. IOSSPOF — Check for single points of failure

Description

The IOSSPOF macro is used to check for I/O configuration redundancy of DASD devices or pairs of DASD devices. To do this IOSSPOF verifies that there are redundant hardware components such that given failure of a hardware component the availability of the device would be unaffected.

Environment

The requirements for the caller of IOSSPOF are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode</td>
<td>Task mode</td>
</tr>
<tr>
<td>Minimum authorization</td>
<td>Problem state. Any PSW key.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit or 64-bit</td>
</tr>
<tr>
<td></td>
<td>If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>The caller must be enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>The caller may not hold any locks.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the IOSSPOF macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if the return code is not 0. Otherwise, used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>
When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**Syntax**

The standard form of the IOSSPOF macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSSPOF.</td>
</tr>
<tr>
<td>IOSSPOF</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSSPOF.</td>
</tr>
<tr>
<td>[,xlabel]</td>
<td>An optional symbol, starting in column 1, that is the name on the IOSSPOF macro invocation. The name must conform to the rules for an ordinary assembler language symbol. DEFAULT: No name.</td>
</tr>
</tbody>
</table>

PERFORM_CHECK

,DEVN1=xdevn1  
xdevn1: RS-type address or register (2) - (12).

[SCHSET1=xschset1]  
xschset1: RS-type address or register (2) - (12).

[DSN1=xds1]  
xds1: RS-type address or register (2) - (12).
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[.DEVN2=xdevn2]</td>
<td>xdevn2: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[.SCHSET2=xschset2]</td>
<td>xschset2: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[.DSN2=xdsn2]</td>
<td>xdsn2: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,VOLSER1=xvolser1</td>
<td>xvolser1: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[.DSN1=xdsn1]</td>
<td>xdsn1: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,[VOLSER2=xvolser2]</td>
<td>xvolser2: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[,DEVLIST=xdevlist]</td>
<td>xdevlist: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,DEVCOUNT=xdevcount</td>
<td>xdevcount: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[,DSNLIST=xdsnlist]</td>
<td>xdsnlist: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,VOLLIST=xvollist</td>
<td>xvollist: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,VOLCOUNT=xvolcount</td>
<td>xvolcount: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[,DSNLIST=xdsnlist]</td>
<td>xdsnlist: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[.SPOFAREA=xspofarea]</td>
<td>xspofarea: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[.HCMMSG=NO]</td>
<td>Default: NO</td>
</tr>
<tr>
<td>[.HCMMSG=YES]</td>
<td></td>
</tr>
<tr>
<td>[HANDLE=xhandle]</td>
<td>xvolser2: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[,WTO=NO]</td>
<td>Default: NO</td>
</tr>
<tr>
<td>[,WTO=YES]</td>
<td></td>
</tr>
<tr>
<td>[,IND_CHECKS=YES]</td>
<td>Default: YES</td>
</tr>
<tr>
<td>[,IND_CHECKS=NO]</td>
<td></td>
</tr>
<tr>
<td>[,IND_CHECKS=ONLY]</td>
<td></td>
</tr>
<tr>
<td>[,SWITCH_CHECKS=YES]</td>
<td>Default: YES</td>
</tr>
<tr>
<td>[,SWITCH_CHECKS=NO]</td>
<td></td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[.CU_CHECKS=YES]</td>
<td>Default: YES</td>
</tr>
<tr>
<td>[.CU_CHECKS=NO]</td>
<td></td>
</tr>
<tr>
<td>[.RETCODE=retcode]</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[.RSNCODE=rsncode]</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>[.PLISTVER=plistver</td>
<td>IMPLIED_VERSION]</td>
</tr>
<tr>
<td>[.MF=S]</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>[.MF=(L,xmfctrl,xmfattr, 0D)]</td>
<td></td>
</tr>
<tr>
<td>[.MF=(M,xmfctrl,COMPLETE</td>
<td>NOCHECK)]</td>
</tr>
<tr>
<td>[.MF=(E,xmfctrl,COMPLETE</td>
<td>NOCHECK)]</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

**,PERFORM_CHECK**

Perform single point of failure checks. The following is a set of mutually exclusive keys. This set is required; only one key must be specified.

**,DEVN1=xdevn1**

Belongs to a set of mutually exclusive keys. It is the name (RS-type), or address in register (2)-(12), of a halfword input containing the device number of a device to check for single points of failure.

**,SCHSET1=xschset1**

This is the name (RS-type), or address in register (2)-(12), of an optional byte input that contains the subchannel set of the device associated with the device number in DEVN1.

Default: 0 (Subchannel set zero).

**,DSN1=xdsn1**

This is the name (RS-type), or address in register (2)-(12), of an optional 44-character input that contains a data set name or a description of the dataset associated with device specified in DEVN1. The keyword is only used for message generation.

Default: * No dataset will be displayed in any outputed messages.

**,DEVN2=xdevn2**

This is the name (RS-type), or address in register (2)-(12), of an optional halfword input that contains a device number of a device used to verify hardware isolation between the devices specified with DEVN1 and this device.

Default: * Pair checking will not be done.
IOSSPOF macro

\texttt{\textbf{,SCHSET2=xschset2}}
This is the name (RS-type), or address in register (2)-(12), of an optional byte input that contains the subchannel set of the device associated with the device number in DEVN2.
\textbf{Default:} 0 (Subchannel set zero).

\texttt{\textbf{,DSN2=xdsn2}}
This the name (RS-type), or address in register (2)-(12), of an optional 44-character input that contains a data set name or a description of the dataset associated with device specified in DEVN2. The keyword is only used for message generation.
\textbf{Default:} * No dataset will be displayed in any outputed messages.

\texttt{\textbf{,VOLSER1=xvolser1}}
Belongs to a set of mutually exclusive keys. It is the name (RS-type), or address in register (2)-(12), of a 6-character input that contains the VOLSER of the device to check for a single point of failure.

\texttt{\textbf{,DSN1=xdsn1}}
This is the name (RS-type), or address in register (2)-(12), of an optional 44-character input that contains a data set name associated with volume specified in VOLSER1. This keyword is only used for message generation.
\textbf{Default:} *

\texttt{\textbf{,VOLSER2=xvolser2}}
It is the name (RS-type), or address in register (2)-(12), of a 6-character input that contains a VOLSER of a volume used to verify hardware isolation between the volumes specified with VOLSER1 and this volume.
\textbf{Default:} *

\texttt{\textbf{,DSN2=xdsn2}}
This is the name (RS-type), or address in register (2)-(12), of an optional 44-character input that contains a data set name associated with volume specified in VOLSER2. This keyword is used for only message generation.
\textbf{Default:} *

\texttt{\textbf{,DEVLIST=xdevlist}}
It is the name (RS-type), or address in register (2)-(12), of a one-byte input that contains the address to an array of fullwords with byte 1 containing zero, byte 2 containing the subchannel set of the device and bytes containing the subchannel set of the device and bytes and 3 and 4 containing the device number of the device to be checked. For example, 0001DE61 represents a device in subchannel set one with a device number of DE61.

\textbf{Note:} Only individual device checks are performed when DEVLIST is specified.

\texttt{\textbf{,DEVCOUNT=xdevcount}}
This is the name (RS-type), or address in register (2)-(12), of a fullword input that contains the number of devices in the DEVLIST array.

End of group of keys.

\texttt{\textbf{,DSNLIST=xdsnlist}}
This is the name (RS-type), or address in register (2)-(12), of an optional
IOSSPOF macro

4-byte input that contains the address of an array of CL44 elements that contain the dataset names of the devices that correspond to the DEVLIST parameter. This keyword is used for message generation only.

Default: *

,VOLLIST=\texttt{xvollist}
This is the name (RS-type), or address in register (2)-(12), of a one-byte input that contains the address an array of CL6 elements containing the VOLSERs of devices to check for single points of failure.

,VOLCOUNT=\texttt{xvolcount}
This is the name (RS-type), or address in register (2)-(12), of a fullword input containing the number of devices in the VOLLIST array.

End of group of keys.

,DSNLIST=\texttt{xdnlist}
This is the name (RS-type), or address in register (2)-(12), of an optional 4-byte input that contains the address of an array of CL44 elements containing the dataset names of the devices that correspond to the VOLLIST parameter. This information is used for only message generation.

Default: *

This ends the of set of mutually exclusive required keywords.

,SPOFAREA=\texttt{xspofarea}
This is the name (RS-type), or address in register (2)-(12), of an optional 4-byte output that will contain the address that contains the data requested. The data is mapped by IOSDSPOF, and is only valid if the service ended with a 4 or 8 return code. The SPOFAREA is obtained by the service and must be released by the caller using the length and subpool specified in the SPOFAREA. The SPOFAREA may be returned in a subpool that is not associated with the issuing task and thus, the caller must not assume the storage is automatically released when the task ends. If the caller is in a PSW key other than key 0-7 when the IOSSPOF service is invoked, the caller must ensure the SPOFAREA storage is accessed while in a PSW key equal to the key of the calling task.

,HCMG=NO|YES
This is an optional keyword input that specifies whether or not health checker messages should be issued automatically with this service. HCMG=YES without a HANDLE is only valid when running under IBM Health Checker for z/OS.

Default: NO.

,HCMG=NO
Indicates that health checker messages should not be issued.

,HCMG=YES
Indicates that health checker messages should be issued through HZSFMSG. HCMG is only valid when the IOSSPOF service is called from a Health Check running under control of the IBM Health Checker for z/OS.

Default: NO

,HANDLE=\texttt{xhandle}
This is the name (RS-type), or address in register (2)-(12), of an optional 16-character input that specifies a handle (token) that identifies the check.
IOSSPOF macro

This handle is returned through the HANDLE parameter of the HZSADDCK macro for a REMOTE=YES check. HANDLE is required when the service is called from a remote check and is ignored when the service is called from a local check. If IBM Health Checker for z/OS is not running at the time of invocation, then a return code of X’10’ with a reason code of ‘02’ will be returned.

Default: * Health checker messages will be issued as a REMOTE=NO call.

,,WTO=NO|YES
This is an optional keyword input that specifies whether or not WTOs of IOSPFxxxI messages will be issued for this service.

Default: NO.

,,WTO=NO
Indicates that WTOs will not be issued for this service.

,,WTO=YES
Indicates that WTOs will be issued for this service. The IOSPFxxxI messages will be issued with a ROUTCDE=11.

,,IND_CHECKS=YES|NO|ONLY
This is an optional keyword input that specifies whether or not single points of failure for individual devices should be checked. For example, checks that are not comparing two devices for mutual single points of failure should be done. This keyword is ignored if a single device is specified. The specific device checks like the following are performed if YES is specified:
• Check to see if a device has only one path available.
• Check to see if the paths of the device share internal hardware subchannel components.

Default: YES.

,,IND_CHECKS=YES
Indicates that individual device checks should be made. That is, all checks should be made.

,,IND_CHECKS=NO
Indicates that individual device checks should not be made or only pair checks should be made.

,,IND_CHECKS=ONLY
Indicates that only individual device checks should be made or no pair checks should be made.

,,SWITCH_CHECKS=YES|NO
This is an optional keyword input that specifies whether or not to check for switch related single points of failure. It applies to individual and pairs checks. The following specific device checks are performed if YES is specified:
• Check if all online CHPIDs are connected to the same switch.
• Check if all devices are connected to the same switch.

Default: YES.

,,SWITCH_CHECKS=YES
Indicates that switch related checks should be made.

,,SWITCH_CHECK=NO
Indicates that switch related checks should not be made.
IOSSPOF macro

, CU_CHECKS=YES|NO
This is an optional keyword input specifies whether or not to check for control unit related single points of failure. It applies to individual and pair checks. The following specific device checks are performed if YES is specified:

- Check if all devices are in the same DASD logical subsystem (LSS).
- Check if all devices are in the same physical control unit.
- Check if all devices are sharing the same set of control unit interfaces.

This keyword is ignored if a single device is specified. That is, DEVN1 is specified without DEVN2 or VOLSER1 is specified without VOLSER2.

Default: YES.

, CU_CHECKS=YES
  Indicates that control unit related checks should be made.

, CU_CHECKS=NO
  Indicates that control unit related checks should not be made.

, RETCODE=retcode
The name (RS-type) of an optional fullword output variable, or register (2)-(12) or (15), into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR15.

, RSNCODE=xrsncode
The name (RS-type) of an optional fullword output variable into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

, PLISTVER=plistver | MAX | IMPLIED_VERSION
Is an optional byte input decimal value in the "1–1" range that specifies the macro version. PLISTVER is the only key allowed on the list form of MF and determines which parameter list is generated. Note that MAX may be specified instead of a number and will cause the parameter list to be of the largest size currently supported. This size may grow from release to release (thus possibly affecting the amount of storage needed by your program). If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as this will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

Default: IMPLIED_VERSION. When PLISTVER is omitted, the default is the lowest version which allows all of the parameters specified on the invocation to be processed.

, MF=S|L|M|E
An optional keyword input that specifies the macro form.

Default: S.

, MF=S
  Specify the standard form of the macro. The "S" form builds an inline parameter list and generates the macro invocation to transfer control to the service. Full checking for required macro keys is done along with supplying defaults for omitted optional parameters.

, MF=(L,xmfctrl,xmfattr, 0D)
  Specifies the list form of the macro. The "L" form defines an storage area for the parameter list. Only the PLISTVER key can be specified on
IOSSPOF macro

the invocation. All other macro parameters are flagged as errors. If PLISTVER is not specified, the original parameter list definition is used.

\textit{xmfctrl} A required input. It is the name of a storage area for the parameter list.

\textit{xmfattr}
An optional 60-character input string that varies from 1 to 60 characters. Use it to force boundary alignment of the parameter list. Use only 0F or 0D. The default is 0D, which forces the parameter list to a doubleword boundary.

\texttt{MF=\{M, xmfctrl, COMPLETE|NOCHECK\}}
Specifies the modify form of the macro. The "M" form generates code to put the parameters into the parameter list specified by \textit{xmfctrl}.

\textit{xmfctrl} A required input. It is the name (RS-type), or address in register (1)-(12), of a storage area for the parameter list.

\texttt{COMPLETE|NOCHECK}
An optional keyword input that specifies the degree of macro parameter syntax checking.

\textit{Default: COMPLETE.}

\texttt{COMPLETE}
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

\texttt{NOCHECK}
Checking for required macro keys is not done or defaults are not supplied for omitted optional parameters.

\texttt{MF=\{E, xmfctrl, COMPLETE|NOCHECK\}}
Specifies the execute form of the macro. The "E" form generates code to put the parameters into the parameter list specified by \textit{xmfctrl} and invoke the desired service.

\textit{xmfctrl} A required input. It is the name (RS-type), or address in register (1)-(12), of a storage area for the parameter list.

\texttt{COMPLETE|NOCHECK}
An optional keyword input that specifies the degree of macro parameter syntax checking.

\textit{Default: COMPLETE.}

\texttt{COMPLETE}
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

\texttt{NOCHECK}
Checking for required macro keys is not done or defaults are not supplied for omitted optional parameters.

\textbf{ABEND codes}
None.
## Return codes

Return codes, in hexadecimal, from the IOSSPOF macro are as follows:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Equate Symbol: SPOF_RC_Ok</td>
</tr>
<tr>
<td></td>
<td>Meaning: No single points of failure detected.</td>
</tr>
<tr>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>04</td>
<td>Equate Symbol: SPOF_RC_SomeChecksFailed</td>
</tr>
<tr>
<td></td>
<td>Meaning: The service couldn't perform all checks specified, but no single points of failure were detected.</td>
</tr>
<tr>
<td></td>
<td>Action: Some checks may fail due to switch devices not being online at the time of the check. All switch devices must be online to determine if control unit interfaces are single point of failure free.</td>
</tr>
<tr>
<td>08</td>
<td>Equate Symbol: SPOF_RC_SPOFFound</td>
</tr>
<tr>
<td></td>
<td>Meaning: Single points of failure were detected.</td>
</tr>
<tr>
<td></td>
<td>Action: Refer to IOSPFxxxI message for action.</td>
</tr>
</tbody>
</table>

### Hex Reason Code

#### Meaning/Action

<table>
<thead>
<tr>
<th>00</th>
<th>Equate Symbol: SPOF_RSN_AllDevicesFound</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Meaning: While a single point of failure was discovered all devices were found.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>01</th>
<th>Equate Symbol: SPOF_RSN_DeviceNotFound</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Meaning: Single points of failure were detected, and one or more of the devices specified are not found.</td>
</tr>
</tbody>
</table>
### Hexadecimal Reason Code 0C

**Equate Symbol: SPOF_RC_ProgramError**

**Meaning:** Program error.

**Action:** None.

#### Hex Reason Code 01

**Equate Symbol: SPOF_RSN_InvalidParmListVers**

**Meaning:** It was discovered that the macro was invoked with an invalid parameter list.

**Action:** Specify a valid parameter list version.

#### Hex Reason Code 02

**Equate Symbol: SPOF_RSN_InvalidCount**

**Meaning:** The number of devices specified via the DEVCOUNT parameter or volume serial numbers via the VOLCOUNT parameter is invalid.

**Action:** Change DEVCOUNT or VOLCOUNT to be less than 65536 and greater than zero.

#### Hex Reason Code 03

**Equate Symbol: SPOF_RSN_ImproperModes**

**Meaning:** IOSSPOF was invoked in an improper mode.

**Action:** See environment specification for what modes IOSSPOF can be invoked and only invoke in supported modes.

#### Hex Reason Code 04

**Equate Symbol: SPOF_RSN_ImproperDevlistEntry**

**Meaning:** A device in the device list did not match the format ‘000sdddd’ where ‘000s’ is the subchannel set and ‘dddd’ is the device number.

**Action:** Adjust DEVLIST parameter to match the format.

#### Hex Reason Code 05

**Equate Symbol: SPOF_RSN_BadParmListAccess**

**Meaning:** Abend accessing parameter list.

**Action:** Verify that the parameters can be accessed by the invoker's key.

### Hexadecimal Reason Code 10

**Equate Symbol: SPOF_RC_EnvironError**

**Meaning:** Environmental error.

**Action:** None.

#### Hex Reason Code 01

**Equate Symbol: SPOF_RSN_IOSSPOFNotAvail**

**Meaning:** The IOSSPOF service is not available at this time.

**Action:** Wait until the IOSAS address space is available.

#### Hex Reason Code 02

**Equate Symbol: SPOF_RSN_HlthChkerNotAvail**

**Meaning:** The Health Checker environment isn’t available and is available if HCMSG=YES and HANDLE is specified.

**Action:** Start the Health Checker if HCMSG=YES is required.

### Hexadecimal Reason Code 20

**Equate Symbol: SPOF_RC_SystemError**

**Meaning:** System error.

**Action:** None.

---

**Chapter 103. IOSSPOF — Check for single points of failure**
### IOSSPOF macro

#### Return and reason codes

Macro IOSDSPOF provides equate symbols for the return and reason codes.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Always set.</td>
</tr>
<tr>
<td>04</td>
<td>00</td>
<td>Always set.</td>
</tr>
<tr>
<td>08</td>
<td>00</td>
<td>Single points of failure were detected, all devices are found.</td>
</tr>
<tr>
<td>08</td>
<td>01</td>
<td>Single points of failure were detected, and one or more of the devices specified are not found.</td>
</tr>
<tr>
<td>0C</td>
<td>01</td>
<td>Incorrect parameter list version.</td>
</tr>
<tr>
<td>0C</td>
<td>02</td>
<td>The number of devices specified through the DEVCOUNT parameter or volume serial numbers specified through the VOLCOUNT parameter is incorrect.</td>
</tr>
<tr>
<td>0C</td>
<td>03</td>
<td>The caller is in an improper mode when invoked.</td>
</tr>
<tr>
<td>0C</td>
<td>04</td>
<td>A device in the device list does not match the format ‘000sdddd’ where ‘000s’ is subchannel set and ‘dddd’ is the device number.</td>
</tr>
<tr>
<td>0C</td>
<td>05</td>
<td>Abend accessing parameter list.</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>The IOSSPOF service is not available at this time.</td>
</tr>
<tr>
<td>10</td>
<td>02</td>
<td>The Health Checker service is not available at this time.</td>
</tr>
<tr>
<td>20</td>
<td>00</td>
<td>Always set.</td>
</tr>
</tbody>
</table>

#### IOSSPOF - List form

Use the list form of the IOSSPOF macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses to contain the parameters.

#### Syntax

The list form of the IOSSPOF macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSSPOF.</td>
</tr>
<tr>
<td>IOSSPOF</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSSPOF.</td>
</tr>
<tr>
<td>[PLISTVER=xplistver</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
</tbody>
</table>
IOSSPOF macro

### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.MF=(L,xmfctrl,xmfattr, 0D),</code></td>
<td><code>xmfctrl</code>: RS-type name or address in register (2)-(12). <code>xmfattr</code>: Any text up to 60 characters. <strong>Default</strong>: 0D</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the IOSSPOF macro, with the following exception:

`.MF=(L,xmfctrl{,xmfattr|0D})`

- **L** specifies the list form of the macro. The "L" form defines an area to be used for the parameter list. Only the PLISTVER key may be specified on the invocation. All other macro parameters are flagged as errors. If PLISTVER is not specified, the original parameter list definition is used.

- **xmfctrl**
  - This required input is the RS-type name, or address in register (1)-(12), of a storage area for the parameter list.

- **xmfattr|0D**
  - This is an optional 60 character input string which is used to force boundary alignment of the parameter list. Use only 0F or 0D.

  The default is 0D, which forces the parameter list to a doubleword boundary.

### IOSSPOF - Execute form

Use the execute form of the IOSSPOF macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

### Syntax

The execute form of the IOSSPOF macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSSPOF.</td>
</tr>
<tr>
<td>IOSSPOF</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSSPOF.</td>
</tr>
<tr>
<td>[.PLISTVER=xplistver</td>
<td><code>Default</code>: IMPLIED_VERSION</td>
</tr>
<tr>
<td></td>
<td>`</td>
</tr>
<tr>
<td><code>.MF=(L,xmfctrl,xmfattr, 0D),</code></td>
<td><code>xmfctrl</code>: RS-type name or address in register (2)-(12). <code>xmfattr</code>: Any text up to 60 characters. <strong>Default</strong>: 0D</td>
</tr>
</tbody>
</table>
IOSSPOF macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>

**Parameters**

The parameters are explained under the standard form of the IOSSPOF macro, with the following exception:

\[ \text{MF}=(E,\text{xmfctrl}{}\{,\text{COMPLETE}\}) \]

E specifies the execute form of the macro. The "E" form generates code to put the parameters into the parameter list specified by xmfctrl and provides full syntax checking with default setting.

\[ \text{,xmfctrl} \]

This required input is the RS-type name, or address in register (1)–(12), of a storage area for the parameter list.

\[ \text{,COMPLETE} \]

An optional keyword which specifies the degree of macro parameter syntax checking. When complete checking is enabled, required parameters are checked and defaults are supplied for omitted optional parameters.

Default: COMPLETE
Chapter 104. IOSUPFA — Obtain address of the UCB prefix extension segment

Note: The UCBLOOK macro is the preferred programming interface.

Description

The IOSUPFA macro obtains the address of the UCB prefix extension segment. To map the UCB prefix extension segment, use the IOSDUPFX mapping macro.

The IOSUPFA macro provides faster performance than the UCBLOOK macro; however, if the caller uses UCBLOOK to obtain several addresses in the same invocation, UCBLOOK might provide better performance than an IOSUPFA macro and an IOSCMXA macro. The UCBLOOK macro also validates input parameters and provides recovery.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state and any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>The caller may hold locks, but is not required to hold any.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>The input parameter must be in the primary address space. If the caller is disabled, the parameter list must reside in fixed or disabled reference (DREF) storage.</td>
</tr>
</tbody>
</table>

Programming requirements

The caller must pass a valid captured or actual UCB address.

The caller must pin the UCB or otherwise guarantee that the UCB will not be deleted. (If the caller issues a UCBLOOK macro with the PIN parameter to pin the UCB, use the UCBLOOK UCBPXPTR parameter rather than the IOSUPFA macro.)

The caller must supply recovery to handle any unexpected errors, such as abends.

Restrictions

None.

Input register information

Before issuing the IOSUPFA macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:
IOSUPFA macro

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Return address</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

**Performance implications**

None.

**Syntax**

The standard form of the IOSUPFA macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name</em>: symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSUPFA.</td>
</tr>
<tr>
<td>IOSUPFA</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSUPFA.</td>
</tr>
<tr>
<td>UCBPTR=ucbptr addr</td>
<td><em>ucbptr addr</em>: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,UCBPADDR=ucbpadr addr</td>
<td><em>ucbpadr addr</em>: RX-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

**UCBPTR=ucbptr addr**

Specifies the address of a fullword field that contains the address of the UCB common segment. This address must not be associated with a copy of the UCB.

**\,UCBPADDR=ucbpadr addr**

Specifies the address of a fullword field in which the system returns the
address of the UCB prefix extension segment. Use the IOSDUPFX mapping macro to map the UCB prefix extension segment.

**ABEND codes**

None.

**Return and reason codes**

None.

### IOSUPFA - List form

Use the list form of the IOSUPFA macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros.

### Syntax

The list form of the IOSUPFA macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSUPFA.</td>
</tr>
<tr>
<td>IOSUPFA</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSUPFA.</td>
</tr>
<tr>
<td>MF=(L,list addr)</td>
<td>list addr: symbol.</td>
</tr>
<tr>
<td>MF=(L,list addr,attr)</td>
<td>attr: 1- to 60-character input string.</td>
</tr>
<tr>
<td>MF=(L,list addr,0D)</td>
<td>Default: 0D</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the IOSUPFA macro with the following exception:

- **MF=(L,list addr)**
- **MF=(L,list addr,attr)**
- **MF=(L,list addr,0D)**

Specifies the list form of the IOSUPFA macro.

*list addr* is the name of a storage area to contain the parameters.

*attr* is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to
force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

**IOSUPFA - Execute form**

Use the execute form of the IOSUPFA macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

**Syntax**

The execute form of the IOSUPFA macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSUPFA.</td>
</tr>
<tr>
<td>IOSUPFA</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSUPFA.</td>
</tr>
<tr>
<td>UCBPTR=ucbptr addr</td>
<td>ucbptr addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_UCBPADDR=ucbpaddr addr</td>
<td>ucbpaddr addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>_MF=(E,list addr)</td>
<td>list addr: RX-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_MF=(E,list addr,COMPLETE)</td>
<td>Default: COMPLETE</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained under the standard form of the IOSUPFA macro with the following exception:

- _MF=(E,list addr)
- _MF=(E,list addr,COMPLETE)

Specifies the execute form of the IOSUPFA macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.
Chapter 105. IOSUPFR — Obtain address of the UCB prefix extension segment

Description

Use the IOSUPFR macro to obtain the address of the UCB prefix extension segment. To map the UCB prefix extension segment, use the IOSDUPFX mapping macro.

UCBLOOK and IOSUPFA macros also provide this function. However, IOSUPFR provides an alternative for passing parameters (that is, in general purpose register (GPR) 1 rather than in a parameter list). For guidance about obtaining UCB information, see z/OS MVS Programming: Authorized Assembler Services Guide.

Environment

The requirements for the caller are:

- **Environmental factor**: Problem state and any PSW key.
- **Minimum authorization**: Task or SRB
- **Dispatchable unit mode**: Any PASN, any HASN, any SASN
- **AMODE**: 24- or 31-bit
- **ASC mode**: Primary
- **Interrupt status**: Enabled or disabled for I/O and external interrupts.
- **Locks**: The caller may hold locks, but is not required to hold any.
- **Control parameters**: None

Programming requirements

The caller must pass a valid captured or actual UCB address.

The caller must pin the UCB or otherwise guarantee that the UCB will not be dynamically deleted.

The caller must supply recovery to handle any unexpected errors, such as abends.

Restrictions

None.

Input register information

Before issuing the IOSUPFR macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of UCB common segment</td>
</tr>
</tbody>
</table>

Before issuing the IOSUPFR macro, the caller does not have to place any information into any access register (AR).
**Output register information**

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>Address of the UCB prefix extension</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**Syntax**

The standard form of the IOSUPFR macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSUPFR.</td>
</tr>
<tr>
<td>IOSUPFR</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSUPFR.</td>
</tr>
<tr>
<td>MF=(S)</td>
<td>Default: S</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

**MF=(S)**

Specifies the standard form of the macro. This parameter is optional.

**ABEND codes**

None.
Return and reason codes
None.
Chapter 106. IOSVRYSW — Vary switch service

Description

IOSVRYSW provides an interface to the VARY SWITCH process to configure a switch port online or offline to Dynamic Channel Path Management (DCM). Invoking this interface for a switch port also causes the specific managed device paths to be varied online or offline. An online request causes the managed channel paths to become eligible to DCM. An offline request causes the managed channel paths to be removed from the control units connected to the managed channel path IDs (CHPID) at the specified ports. This command affects only managed device paths. Non-managed paths must be varied online or offline separately.

Note: VARY SWITCH command performs the same function when it is issued from a console. For more information, see Placing a Switch Port Online or Offline in z/OS MVS System Commands.

Macro IOSDVSAP maps each element of the array of resource elements that is passed to the VARY SWITCH programming interface. Each element is created by a separate IOSVRYSW BUILD invocation and represents a vary switch port online, offline, or offline and unconditional request.

Note: The caller is responsible for obtaining the right amount of storage for the array of resource elements before the first IOSVRYSW BUILD request.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state, or PSW key 0-7, or APF-authorized</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space</td>
</tr>
</tbody>
</table>

Programming requirements

Programs invoking this interface must include mapping macro IOSDVSAP.

Restrictions

None.

Input register information

Before issuing the IOSVRYSW macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Register

Contents
**IOSUPFR macro**

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>undefined</td>
</tr>
<tr>
<td>1</td>
<td>used by the service</td>
</tr>
<tr>
<td>2-13</td>
<td>undefined</td>
</tr>
<tr>
<td>14-15</td>
<td>used by the service</td>
</tr>
</tbody>
</table>

**Output register information**

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reason code (valid for TYPE=INVOK, unpredictable otherwise)</td>
</tr>
<tr>
<td>1</td>
<td>unpredictable</td>
</tr>
<tr>
<td>2-13</td>
<td>unchanged</td>
</tr>
<tr>
<td>14</td>
<td>unpredictable</td>
</tr>
<tr>
<td>15</td>
<td>return code (valid for TYPE=INVOK, unpredictable otherwise)</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>unchanged</td>
</tr>
</tbody>
</table>

**Performance implications**

None.

**Syntax**

The standard form of the IOSVRYSW macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>one or more blanks must precede IOSVRYSW.</td>
</tr>
</tbody>
</table>

IOSVRYSW

| b                       | one or more blanks must follow IOSVRYSW.                                    |

TYPE=BUILD

| ,REQUEST=ONLINE         | default: ONLINE                                                              |
| ,REQUEST=OFFLINE        |                                                                              |

| ,OPTION=UNCOND          | default: NONE                                                                |
**Parameters**

The parameters are explained as follows:

**TYPE=BUILD**

Specifies a required keyword input which indicates that the macro is being invoked to construct vary switch parameters.

**REQUEST=ONLINE**

REQUEST=OFFLINE

Specifies an optional keyword input which indicates the type of request to process.

- **ONLINE**: The request is to configure a switch port online to Dynamic Channel Path Management. The default is **ONLINE**.
- **OFFLINE**: The request is to configure a switch port offline to Dynamic Channel Path Management.

**OPTION=UNCOND**

Specifies an optional keyword input which indicates an additional option to be processed along with an OFFLINE request.

**UNCOND**: This option is used to specify an UNCOND request on the VARY PATH commands invoked as a result of the VARY SWITCH request. Adding the UNCOND keyword to a VARY PATH,OFFLINE command results in the system taking offline the last path to devices that are online but unallocated.
**SWITCHDEV**=*switchdev*

Specifies an RS-type name, or address in register (2)-(12), of a required halfword input which indicates the switch device number to be affected.

**PORTADDR**=*portaddr*

Specifies an RS-type name, or address in register (2)-(12), of a required byte input which indicates the port address to be affected.

**TYPE**=*INVOKE*

Specifies a required keyword input which indicates to perform the requested Vary Switch function built by one or multiple IOSVRYSW BUILD requests.

**DATAADDR**=*dataaddr*

Specifies an RS-type name, or address in register (2)-(12), of a required 4 byte input that contains the address to the array of resource elements to be processed. Each element is created by an IOSVRYSW BUILD invocation and is mapped by mapping macro IOSDVSAP.

**DATANUM**=*datanum*

Specifies an RS-type name, or address in register (2)-(12), of a required fullword input that contains the number of elements in the array of resource elements to be processed.

**RETCODE**=*retcode*

Specifies an RS-type name of an optional fullword output variable, or register (2)-(12), into which the return code is to be copied from GPR 15.

**Note:** This keyword can only be specified with an INVOKE request.

**RSNCODE**=*rsncode*

Specifies an RS-type name of an optional fullword output variable, or register (2)-(12), into which the reason code is to be copied from GPR 0.

**Note:** This keyword can only be specified with an INVOKE request.

**MF**=(*E*, list addr)

**MF**=(*E*, list addr, COMPLETE)

Specifies the execute form of the IOSVRYSW macro.

`list addr`

Specifies the area that the system uses to contain the parameters.

**COMPLETE**

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters. COMPLETE is the default.

**ABEND codes**

None.

**Return and reason codes**

**Note:** There are no return or reason codes for TYPE=BUILD.

When the system returns control to the caller, GPR 15 (and `retcode`, when you code RETCODE) contains a return code.

The following table identifies the hexadecimal return codes:
Table 91. Return Codes for the IOSVRYSW Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | Meaning: Successful completion.  
Action: None required. |
| 10                      | Meaning: An unexpected error occurred in vary switch processing.  
Action: Verify the configuration in effect and resubmit the request. If the request fails again for the same reason, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
| FF04                    | Meaning: Storage passed on the DATAADDR was not accessible by the service.  
Action: Verify that accessible storage is being passed. |
| FF08                    | Meaning: The attempt to queue a work element to the IOS address space failed. Request is currently not able to be performed.  
Action: Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
| FF0C                    | Meaning: VSAP data is readable but not valid.  
Action: Verify that the correct data is being passed. |
| FF10                    | Meaning: Caller is not in a valid environment to invoke the IOSVRYSW API.  
Action: Insure that the caller is running in the correct environment. |
| FF14                    | Meaning: Module IOSVVSFWF suffered a catastrophic error. Function could not be processed.  
Action: Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
| FF18                    | Meaning: Module IOSVVSFWF could not establish a recovery environment.  
Action: Resubmit the request. If the request fails again for the same reason, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. |

Hexadecimale Reason Codes: None.

**IOSVRYSW—List form**

Use the list form of the IOSVRYSW macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

**Syntax**
The list form of the IOSVRYSW macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSVRYSW.</td>
</tr>
</tbody>
</table>

Chapter 106. IOSVRYSW — Vary switch service  1075
**IOSUPFR macro**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSVRYSW.</td>
</tr>
<tr>
<td>MF=(L, list addr)</td>
<td>list addr: Symbol.</td>
</tr>
<tr>
<td>MF=(L, list addr, attr)</td>
<td>attr: 1- to 60-character input string</td>
</tr>
<tr>
<td>MF=(L, list addr, 0D)</td>
<td>Default: 0D</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

- **MF=(L, list addr)**
- **MF=(L, list addr, attr)**
- **MF=(L, list addr, 0D)**

Specifies the list form of the IOSVRYSW macro.

- list addr is the name of a storage area to contain the parameters.

- attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

**Examples**

An example of issuing three IOSVRYSW TYPE=BUILD and an IOSVRYSW TYPE=INVOKE invocations to process three switch ports.

**Initial Setup:**

- Define the list form of the macro
- Obtain storage for the array of resource elements (See mapping macro IOSDVSAP in z/OS MVS Data Areas in the z/OS Internet library)
- Establish addressability to the area
- Clear the area
- Set up a pointer, pointing to the beginning of the area
- Base VSAP_RESOURCE structure on the pointer that points to the beginning of this area.
- For EACH port to be altered (for each request), issue IOSVRYSW TYPE=BUILD:
  - IOSVRYSW TYPE=BUILD,
    REQUEST=ONLINE,
    SWITCHDEV=switch
    PORTADDR=port_address
  - Advance the pointer by length of VSAP_RESOURCE to the next slot in the array
  - IOSVRYSW TYPE=BUILD,
    REQUEST=ONLINE,
    SWITCHDEV=switch
    PORTADDR=port_address
  - Advance the pointer by length of VSAP_RESOURCE to the next slot in the array
- IOSVRYSW TYPE=BUILD,
  REQUEST=ONLINE,
  SWITCHDEV=switch
  PORTADDR=port_address
- Issue IOSVRYSW TYPE=INVOKE to process the requests, passing in the
  pointer to the array of resource elements and the number of elements to the
  processing module.
- IOSVRYSW TYPE=INVOKE,
  DATAADDR=pointer to the array of resource elements,
  DATANUM=number of elements to be processed
  (ports to be altered),
  RETCODE=RETURN_CODE,
  RSNCODE=REASON_CODE,
  MF=(E, IOSVRYSW_LIST)
IOSUPFR macro
Chapter 107. IOSWITCH — IOS switch information service

Description

IOSWITCH provides a service which callers outside the IOS address space can use to obtain physical topology information about a specific switch and its ports.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem or Supervisor state. Any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks may be held.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

The invoker must not hold any locks which would prevent this service from obtaining the IOSYNCH lock. The service must not be invoked until after the IOS space-switching PC table has been established.

Input register information

Before issuing the IOSWITCH macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

The contents of registers 14 through 1 are altered during processing.

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR15 is not 0</td>
</tr>
<tr>
<td>1</td>
<td>Unpredictable (Used as a work register by the system)</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Unpredictable (Used as a work register by the system)</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:
**IOSWITCH macro**

**Register**

**Contents**

0-1  Unpredictable (Used as work registers by the system)
2-13 Unchanged
14-15 Unpredictable (Used as work registers by the system)

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**Syntax**

The IOSWITCH macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><code>name</code>: Symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td></td>
<td>One or more blanks must precede IOSWITCH.</td>
</tr>
<tr>
<td>IOSWITCH</td>
<td>One or more blanks must follow IOSWITCH.</td>
</tr>
<tr>
<td>SWITCH=switch</td>
<td><code>switch</code>: Symbol up to 4 characters long.</td>
</tr>
<tr>
<td>SWITCHAREA=switcharea</td>
<td><code>switcharea</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>SWITCHLEN=switchlen</td>
<td><code>switchlen</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>SUBPOOL=subpool</td>
<td><code>subpool</code>: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>OFFLINE</td>
<td></td>
</tr>
<tr>
<td>RETCODE=rcode</td>
<td><code>rcode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>RSNCODE=rsncode</td>
<td><code>rsncode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>PLISTVER=1</td>
<td></td>
</tr>
</tbody>
</table>
### Parameters

The parameters are explained as follows:

- **name**
  
  An optional symbol, starting in column 1, that is the name on the IOSWITCH macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

- **SWITCH=switch**
  
  A required 4-character input parameter containing the switch device number.

- **SWITCHAREA=switcharea**
  
  A required 4-byte output parameter that will receive the address of the switch data area. The storage for this data must be released by the caller. This data is mapped by IOSDSWTD.

- **SWITCHLEN=switchlen**
  
  A required fullword output parameter that will receive the length, in bytes, of the switch data area.

- **SUBPOOL=subpool**
  
  The name (RS-type), or address in register (2) - (12), of a required halfword input parameter that identifies the subpool to be used for obtaining storage for the switch data area.

  When the calling program is unauthorized, storage is obtained in the specified subpool, provided that the caller is permitted to use that subpool. Storage will be obtained in the caller’s key; however, the resulting key will be set according to the rules for the specified subpool, as documented in *z/OS MVS Programming: Assembler Services Guide*. Valid subpools are: 0, 127, 131, and 132.

  When the calling program is authorized, storage is obtained in key 0. Valid subpools are: 226, 227, 228, 231, 239, 241, 245, 247, and 248.

- **OFFLINE**
  
  An optional keyword that indicates that data will be returned for the switch device even if the device is offline. Note that if the device is in fact offline, the data may be outdated.

- **RETCODE=retcode**
  
  An optional output parameter into which the return code is to be copied from GPR 15.

### To code

Specify the RS-type address of a fullword field, or register (2)-(12).
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
  
  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.
**list addr**
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

**attr**
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

**COMPLETE**
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

**ABEND codes**
None.

**Return and reason codes**
When the IOSWITCH macro returns control to your program:
- GPR 15 (and retcode, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes:

<table>
<thead>
<tr>
<th>Hexadecimal return code</th>
<th>Reason codes, meaning and action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>IOSWITCH completed successfully.</td>
</tr>
<tr>
<td>04</td>
<td>Warning.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reason code</th>
<th>Meaning/Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Meaning: The switch device provided by the caller is offline. No data is returned. Action: To obtain data for the offline switch, use the OFFLINE keyword.</td>
</tr>
<tr>
<td>02</td>
<td>Meaning: The IOSWITCH service is not enabled at this time. No data is returned. Action: Try the service again at a later time.</td>
</tr>
<tr>
<td>03</td>
<td>Meaning: The switch table is not available. No data is returned. Action: Check dynamic channel path management status, as it pertains to the switch table availability.</td>
</tr>
<tr>
<td>Hexadecimal return code</td>
<td>Reason codes, meaning and action</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>08</td>
<td>Program error.</td>
</tr>
<tr>
<td><strong>Reason code</strong></td>
<td><strong>Meaning/action</strong></td>
</tr>
<tr>
<td>01</td>
<td>Meaning: An authorized calling program specified an unauthorized subpool or an unsupported authorized subpool.</td>
</tr>
<tr>
<td></td>
<td>Action: Correct the subpool and reissue the IOSWITCH macro. Authorized programs are restricted to subpools 226, 227, 228, 231, 239, 241, 245, 247, and 248.</td>
</tr>
<tr>
<td>02</td>
<td>Meaning: The switch device number provided by the caller is not in the switch table.</td>
</tr>
<tr>
<td></td>
<td>Action: Correct the switch device number and reissue the IOSWITCH macro.</td>
</tr>
<tr>
<td>03</td>
<td>Meaning: Program error. An error occurred in accessing the caller’s parameter list.</td>
</tr>
<tr>
<td></td>
<td>Action: Ensure that the storage area for the parameter list is addressable in the caller’s primary address space using the key of the caller.</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: An unauthorized calling program specified an authorized subpool or an unsupported unauthorized subpool.</td>
</tr>
<tr>
<td></td>
<td>Action: Correct the subpool and reissue the IOSWITCH macro. Unauthorized programs are restricted to subpools 0 - 127, 131, and 132.</td>
</tr>
<tr>
<td>20</td>
<td>System error. An unexpected error occurred.</td>
</tr>
</tbody>
</table>
Chapter 108. IOSZHPF — zHPF channel program capabilities service

Description

The IOSZHPF macro provides information about the zHPF capabilities of a device from the operating system, processor, channel, and device point of view. The information returned is mapped by IOSDZHPF and reflects the minimum capability of all of the online channels for a device. For example, if a device has two online channels and one channel supports a new function and the other does not, that function will not be reported.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem or supervisor state. Any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>The caller may hold locks, but is not required to hold any.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>If the caller is disabled, the parameter list must reside in fixed or disabled reference (DREF) storage.</td>
</tr>
</tbody>
</table>

Programming requirements

Users of this macro must make sure that the UCB will not be deleted. The application must pin the UCB, or make sure that the environment it is executing in, will not allow the UCB to be deleted.

This service will not have any recovery. The user must supply recovery to handle any unexpected errors.

Restrictions

None.

Input register information

Before issuing the IOSZHPF macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a 144 byte save area</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**IOSZHPF macro**

0  Reason code
1  Used as a work register by the system
2-13 Unchanged
14 Used as a work register by the system
15 Return code

**Performance implications**

None.

**Syntax**

The standard form of the IOSZHPF macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>name:</em> Symbol. Begin <em>name</em> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IOSZHPF.</td>
</tr>
<tr>
<td>IOSZHPF</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IOSZHPF.</td>
</tr>
<tr>
<td>INFOAREA=infoarea</td>
<td><em>infoarea:</em> RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,UCBPTR=ucbptr</td>
<td><em>ucbptr:</em> RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,DEVINFO=NO</td>
<td><strong>Default:</strong> NO</td>
</tr>
<tr>
<td>,DEVINFO=YES</td>
<td></td>
</tr>
<tr>
<td>,LINKAGE=BRANCH</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=xretcode</td>
<td><em>xretcode:</em> RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=xrsncode</td>
<td><em>xrsncode:</em> RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=[</td>
<td>xplistver [MAX</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:
IOSZHPF macro

name
An optional symbol, starting in column 1, that is the name on the IOSZHPF macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

INFOAREA=infoarea
The name (RS-type), or address in register (2)-(12), of a required 32 character input into which IOSZHPF is to return the zHPF channel program information. This area is mapped by IOSDZHPF.

,UCBPTR=ucbptr
The name (RS-type), or address in register (2)-(12), of a required 4 byte input that contains the address of the UCB (common segment address mapped by IEFUCBOB) whose zHPF information is to be obtained.

,DEVINFO=NO
,DEVINFO=YES
Specifies whether the device-related zHPF capabilities should be returned.

NO The device-related zHPF capabilities should not be returned. NO is the default.

YES The device-related zHPF capabilities should be returned.

,LINKAGE=BRANCH
An optional keyword input that indicates that branch-entry linkage should be issued for the routine invocation.

,RETCODE=xretcode
The name (RS-type), or register (2)-(12), of an optional fullword output into which the return code is to be copied from GPR 15.

,RSNCODE=xrsncode
The name (RS-type), or register (2)-(12), of an optional fullword output into which the reason code is to be copied from GPR 0.

,PLISTVER=xplistver | MAX | IMPLIED_VERSION
An optional byte input decimal value in the "1-1" range that specifies the macro version. PLISTVER is the only key allowed on the list form of MF and determines which parameter list is generated. Note that MAX may be specified instead of a number and will cause the parameter list to be of the largest size currently supported. This size may grow from release to release (thus possibly affecting the amount of storage needed by your program). If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as this will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

Default: IMPLIED_VERSION. When PLISTVER is omitted, the default is the lowest version which allows all of the parameters specified on the invocation to be processed.

ABEND codes
None.

Return and reason codes
When the IOSZHPF macro returns control to your program:

- GPR 15 (and xretcode when you code RETCODE) contains a return code.
When the value in GPR 15 is not zero, GPR 0 (and `xrsncode` when you code `RSNCODE`) contains a reason code.

The following table identifies the hexadecimal return and reason codes:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Successful completion</td>
</tr>
<tr>
<td>04</td>
<td>Warning</td>
</tr>
<tr>
<td></td>
<td>Reason Code</td>
</tr>
<tr>
<td></td>
<td>Meaning</td>
</tr>
<tr>
<td>01</td>
<td>zHPF is disabled for the device.</td>
</tr>
<tr>
<td>08</td>
<td>Program error</td>
</tr>
<tr>
<td></td>
<td>Reason Code</td>
</tr>
<tr>
<td></td>
<td>Meaning</td>
</tr>
<tr>
<td>01</td>
<td>The UCB address provided by the caller does not represent a valid UCB.</td>
</tr>
</tbody>
</table>
Chapter 109. IQPINFO — Obtain PCIe information

Description

The IQPINFO macro provides PCIe-related information, including any performance statistics.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state and PSW key 0</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB mode</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller may hold locks, but is not required to hold any.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the ENQ macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller of the IQPINFO macro, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unchanged</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller of the IQPINFO macro, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IQPINFO macro

0-1  Used as work registers by the system
2-13 Unchanged
14-15 Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service and restore them after the system returns control.

Performance implications
None.

Syntax
The IQPINFO macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IQPINFO.</td>
</tr>
<tr>
<td>IQPINFO</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IQPINFO.</td>
</tr>
<tr>
<td>PERFDATA</td>
<td></td>
</tr>
<tr>
<td>_FCNINDEX=xfcnindex</td>
<td>xfcnindex: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_PERFDATAPTR=xperfdataptr</td>
<td>xperfdataptr: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_PERFDATALEN=xperfdataalen</td>
<td>xperfdataalen: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td>_LINKAGE=BRANCH</td>
<td>Default: BRANCH</td>
</tr>
<tr>
<td>_RETCODE=xretnode</td>
<td>xretnode: RS-type address or address in register (2) - (12) or (15).</td>
</tr>
<tr>
<td>_RSNCODE=xrsncode</td>
<td>xrsncode: RS-type address.</td>
</tr>
<tr>
<td>_PLISTVER=xplistver</td>
<td></td>
</tr>
<tr>
<td>_PLISTVER=IMPLIED_VERSION</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>_MF=S</td>
<td>Default: MF=S</td>
</tr>
</tbody>
</table>
| _MF=(L,xmfctr,xmfatrr | 0D)
Parameters

The parameters are explained as follows:

**PERFDATA**
Obtains PCIe-related performance information.

**,FCNINDEX=xfcnindex**
Specifies the RS-type address or address in register (2) - (12) of a required fullword input of PCIe function index to start the search from. The IQPINFO service returns performance data for the next function in the PCIe function table after the supplied index value.

If an allocated PCIe function is found in the table after the supplied PCIe function index, the XFCNINDEX field is set to the index of the PCIe function upon which the performance data is returned. If an allocated PCIe function is not found after the supplied index, the XFCNINDEX field is set to 0. To get information for all allocated PCIe functions, this service should be called continuously starting with a 0 function index until a 0 function index is returned.

**,PERFDATAPTR=xperfdataptr**
Specifies the RS-type address or address in register (2) - (12) of a required 8-byte input that contains the address of an area which is to receive the performance data. This area must be addressable in the primary address space. This area is mapped by the IQPYPERF mapping macro.

**,PERFDATALEN=xperfdatalen**
Specifies the RS-type address or address in register (2) - (12) of a required fullword input that contains the length of the area to receive the performance data. If the supplied length is not large enough to contain the data to be returned, the supplied xperfdatalen field is set to the length required to fit all of the data to be returned.

**,LINKAGE=BRANCH**
An optional keyword input that indicates the linkage that should be generated for the routine invocation. The default is LINKAGE=BRANCH.

LINKAGE=BRANCH requests branch-entry invocation.

**,RETCODE=xretcode**
The name (RS-type) of an optional full-word output variable, or register (2)-(12) or (15), into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

**,RSNCODE=xrsncode**
The name (RS-type) of an optional fullword output variable into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

**,PLISTVER=xplistver**
An optional byte input decimal value (with a value of 1) that specifies the macro version. PLISTVER is the only key allowed on the list form of MF and IQPINFO macro.
determines which parameter list is generated. Note that MAX may be specified
instead of a number, and the parameter list will be of the largest size currently
supported. This size may grow from release to release (thus possibly affecting
the amount of storage needed by your program). If your program can tolerate
this, IBM recommends that you always specify MAX when creating the list
form parameter list as that will ensure that the list form parameter list is
always long enough to hold whatever parameters might be specified on the
execute form.

The default is IMPLIED_VERSION. When PLISTVER is omitted, the default is
the lowest version which allows all of the parameters specified on the
invocation to be processed.

\[ MF=S \]
\[ MF=(L,xmfctrl,xmfattr | 0D) \]
\[ MF=(E,xmfctrl,COMPLETE) \]

An optional keyword input which specifies the macro form. The default is S.

\[ MF=S \]

Specifies the standard form of the macro. The 'S' form generates code to
put the parameters into an in-line parameter list and invoke the desired
service. Full checking for required macro keys is done along with
supplying defaults for omitted optional parameters.

\[ MF=(L,xmfctrl,xmfattr | 0D) \]

Specifies the list form of the macro. The 'L' form defines an area to be used
for the parameter list. Only the PLISTVER key may be specified on the
invocation. All other macro parameters are flagged as errors. If PLISTVER
is not specified, the original parameter list definition is used.

\[ xmfctrl \]

A required input. It is the name of a storage area for the parameter list.

\[ xmfattr | 0D \]

An optional 60-character input string that varies from 1 to 60
characters. Use it to force boundary alignment of the parameter list.
Use only 0F or 0D. The default is 0D, which forces the parameter list to
a doubleword boundary.

\[ MF=(E,xmfctrl,COMPLETE) \]

Specifies the execute form of the macro. The 'E' form generates code to put
the parameters into the parameter list specified by xmfctrl and provides
full syntax checking with default setting.

\[ xmfctrl \]

A required input. It is the name (RS-type) or address in register (1) -
(12) of a storage area for the parameter list.

\[ COMPLETE \]

An optional keyword input which specifies the degree of macro
parameter syntax checking. The default is COMPLETE.

### Return codes

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | **Meaning**: The requested function was successfully executed.  
|                         | **Action**: None.                           |
### IQPINFO macro

**Table 93. Return codes for the IQPINFO Macro (continued)**

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>The supplied data area that receives the output from the IQPINFO macro is not large enough to hold all of the data to be returned. The supplied length field is set to contain the size of the area that is required to fit all of the data to be returned.</td>
<td>Increase the size of the supplied data area to be at least the size that is returned in the PERFDATALEN parameter and retry the IQPINFO request.</td>
</tr>
<tr>
<td>0C</td>
<td>Caller is not authorized to use the IQPINFO macro.</td>
<td>The caller of IQPINFO must have proper authorization (supervisor state and running with PSW key 0).</td>
</tr>
<tr>
<td>10</td>
<td>Unexpected error occurred during IQPINFO processing.</td>
<td>Retry the IQPINFO request. If the problem persists, record the return codes and supply them to the appropriate IBM support personnel.</td>
</tr>
<tr>
<td>14</td>
<td>The requested IQPINFO PLISTVER number is incorrect.</td>
<td>Specify the correct PLISTVER number in the invocation of the IQPINFO macro. If PLISTVER(MAX) is specified, the parameter list will be of the largest size currently supported.</td>
</tr>
<tr>
<td>18</td>
<td>The supplied data area address is not on a doubleword boundary.</td>
<td>Provide a data area with an address that is on a 8 byte boundary (that is, the last 3 bits of the address must be 0).</td>
</tr>
<tr>
<td>20</td>
<td>The IQPINFO service is not available. PCIE services have not been enabled on the current system.</td>
<td>Report the problem to the system programmer.</td>
</tr>
</tbody>
</table>
IQPINFO macro
Chapter 110. IRDFSD — FICON switch data services

Description

The FICON Switch Data macro service is used to obtain statistical counters from FICON switch devices.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem or Supervisor state or any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit addressing mode.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

None

Restrictions

- No locks can be held
- Must not be in an environment that would prevent EXCP from being issued
- Must be authorized

Input register information

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

Before issuing the IRDFSD macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason Code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Restored</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>
IRDFSD macro

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IRDFSD macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IRDFSD.</td>
</tr>
<tr>
<td>IRDFSD</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IRDFSD.</td>
</tr>
<tr>
<td>DEVICE=ALL</td>
<td>Default: DEVICE=ALL</td>
</tr>
<tr>
<td>DEVICE=SINGLE</td>
<td></td>
</tr>
<tr>
<td>,DEVICENUMBER=devicenumber</td>
<td>Required with DEVICE=SINGLE</td>
</tr>
<tr>
<td>,COUNTERS=DEFAULT</td>
<td>Default: COUNTERS=DEFAULT</td>
</tr>
<tr>
<td>COUNTERS=ALL</td>
<td></td>
</tr>
<tr>
<td>COUNTERS=LIST</td>
<td></td>
</tr>
<tr>
<td>,COUNTERLIST=counterlist</td>
<td>Required with COUNTER=LIST</td>
</tr>
<tr>
<td>SUBPOOL=0</td>
<td>Default: SUBPOOL=0</td>
</tr>
<tr>
<td>SUBPOOL=subpool</td>
<td></td>
</tr>
<tr>
<td>FSDADDRESS=fsdaddress</td>
<td></td>
</tr>
<tr>
<td>FSDLENGTH=fsdlength</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=1</td>
<td></td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.MF=S</code></td>
<td>Default: MF=S</td>
</tr>
<tr>
<td><code>.MF=(L,mfctrl)</code></td>
<td>list addr: RS-type address or register (1) - (12).</td>
</tr>
<tr>
<td><code>.MF=(L,</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(E,</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(E,</code></td>
<td></td>
</tr>
<tr>
<td><code>.MF=(E,</code></td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

- **name**
  
  An optional symbol, starting in column 1, that is the name on the IRDFSD macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

- **DEVICE=ALL**
  **DEVICE=SINGLE**
  
  An optional parameter that indicates the one or more devices for which statistics should be returned.

  - **DEVICE=ALL**
    
    Return statistics for all devices.

  - **DEVICE=SINGLE**
    
    Return statistics for a single device.

- **DEVICENUMBER=devicenumber**
  
  A required input parameter if DEVICE=SINGLE is specified. It identifies the device number of the FICON switch device to be interrogated.

  To code: Specify the RS-type address or address in register (2)-(12), of a 4 byte field.

- **COUNTERS=DEFAULT**
  **COUNTERS=ALL**
  **COUNTERS=LIST**
  
  An optional parameter that indicates the set of counters to be returned.

  - **COUNTERS=DEFAULT**
    
    Specifies that the default set of counters is to be returned.

  - **COUNTERS=ALL**
    
    Specifies that all supported counters should be returned.

  - **COUNTERS=LIST**
    
    Specifies that the list of counters to be returned has been supplied.

  - **COUNTERLIST=counterlist**
    
    A required input parameter if COUNTER=LIST is specified. It contains the address that specifies a list of Statistical Counter Control Words to return.

  To code: Specify the RS-type address or address in register (2)-(12), of a 4 byte field.
IRDFSD macro

The number of Control Words cannot exceed 60.

See the IHAFSD macro for further information about the definition of the statistical counter list (SCCW) and the counters available.

\[ \text{\texttt{SUBPOOL=0}} \]

**Subpool=**

A optional input parameter that specifies the subpool to be used to obtain the storage for the FICON Switch Dat (FSD). The default is 0.

**To code:** Specify the RS-type address or address in register (2)-(12) of the 1 byte field containing the subpool.

The returned information is mapped in macro IHAFSD.

\[ \text{\texttt{FSDADDRESS=fsdaddress}} \]

A required input parameter which contains the address of the storage mapped in IHAFSD.

The mapping macro IHAFSD can be found in SYS1.MODGEN.

**To code:** Specify the RS-type address of a fullword field, or register (2)-(12).

\[ \text{\texttt{FSDLNGTH=fsdlength}} \]

A required input parameter which contains the length of the storage mapped in IHAFSD.

**To code:** Specify the RS-type address of a 4-byte field, or register (2)-(12).

\[ \text{\texttt{RETCODE=retcode}} \]

An optional output parameter into which the return code is to be copied from GPR 15.

**To code:** Specify the RS-type address of a fullword field, or register (2)-(12).

\[ \text{\texttt{RSNCODE=rsncode}} \]

An optional output parameter into which the reason code is to be copied from GPR 0.

**To code:** Specify the RS-type address of a fullword field, or register (2)-(12).

\[ \text{\texttt{PLISTVER=IMPLIED_VERSION}} \]

\[ \text{\texttt{PLISTVER=MAX}} \]

\[ \text{\texttt{PLISTVER=1}} \]

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM suggests that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **1**, if you use the currently available parameters.
IRDFSD macro

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

, MF=S
, MF=(L,mfctrl)
, MF=(L,mfctrl,mfattr)
, MF=(L,mfctrl,0D)
, MF=(E,mfctrl)
, MF=(E,mfctrl,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. The execute form of the macro stores the parameters into the storage area defined by xmfctrl and provides full syntax checking with the default setting.

,xmfctrl
This is a required keyword that specifies a storage area for the parameter list. This can be an RS-type address or an address in register (1)-(12).

,xmfattr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

None.

Return codes

When the IRDFSD macro returns control to your program:

- GPR 15 (and retcode, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: Successful completion. Data returned for all FICON switches.</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: No FICON switch devices found. No FSD area was obtained.</td>
</tr>
<tr>
<td>08</td>
<td>Meaning: I/O errors occurred. Some switches did not return data.</td>
</tr>
</tbody>
</table>
Table 94. Return Codes for IRDFSD macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C</td>
<td>Meaning: I/O errors occurred. No data obtained for any devices.</td>
</tr>
<tr>
<td>10</td>
<td>Meaning: Unexpected error.</td>
</tr>
</tbody>
</table>

Reason codes

The following table identifies the hexadecimal return and reason codes:

Table 95. Return and Reason Codes for IRDFSD macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>04</td>
<td>Required module unavailable. The request could not be processed.</td>
</tr>
<tr>
<td>00000002</td>
<td>04</td>
<td>The server task is not active. The request could not be processed.</td>
</tr>
<tr>
<td>00000024</td>
<td>04</td>
<td>FICON switch statistics disabled.</td>
</tr>
<tr>
<td>00000003</td>
<td>08</td>
<td>Caller is not APF authorized.</td>
</tr>
<tr>
<td>00000008</td>
<td>08</td>
<td>IRDVFSD does not recognize the request type.</td>
</tr>
<tr>
<td>00000021</td>
<td>08</td>
<td>Specified device is not a FICON switch.</td>
</tr>
<tr>
<td>00000022</td>
<td>08</td>
<td>Specified device is not online.</td>
</tr>
<tr>
<td>00000023</td>
<td>08</td>
<td>IOSVFSD does not recognize the request type.</td>
</tr>
<tr>
<td>00000004</td>
<td>0C</td>
<td>Recovery could not be established.</td>
</tr>
<tr>
<td>00000005</td>
<td>0C</td>
<td>POST failed.</td>
</tr>
<tr>
<td>00000006</td>
<td>0C</td>
<td>Recovery was entered.</td>
</tr>
<tr>
<td>00000025</td>
<td>0C</td>
<td>I/O error.</td>
</tr>
<tr>
<td>00000026</td>
<td>0C</td>
<td>UCBCSCAN failure.</td>
</tr>
<tr>
<td>00000027</td>
<td>0C</td>
<td>UCBLOOK failure.</td>
</tr>
<tr>
<td>00000028</td>
<td>0C</td>
<td>Server task recovery entered.</td>
</tr>
</tbody>
</table>
Chapter 111. IRDFSDU — FICON switch data update services

Description

The FICON Switch Data macro update service is used to update statistical counters from FICON switch devices.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem or Supervisor state or any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit addressing mode.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

None

Restrictions

- No locks may be held
- Must not be in an environment that would prevent EXCP from being issued
- Must be authorized

Input register information

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–15</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

Before issuing the IRDFSDU macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason Code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Restored</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>
IRDFSDU macro

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications
None.

Syntax
The IRDFSDU macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IRDFSDU.</td>
</tr>
<tr>
<td>IRDFSDU</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IRDFSDU.</td>
</tr>
<tr>
<td>,TOLERANCE=tolerance</td>
<td>tolerance: RS-type address or address in register (2) — (12).</td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td>Default: PLISTVER=IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER[MAX]</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=1</td>
<td></td>
</tr>
<tr>
<td>,MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>,MF=(L,mfctrl)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,mfctrl,mfattr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,mfctrl,0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,mfctrl)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IRDFSDU macro invocation. The name must conform to the rules for an ordinary assembler language symbol.
A required input parameter which contains the input tolerance for up-to-date test of last update performed.

To code: Specify the RS-type address or address in register (2)-(12), of an 8 byte field.

An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- IMPLIED_VERSION, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- MAX, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- 1, if you use the currently available parameters.

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 1

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.
IRDFSDU macro

Use MF=L to specify the list form of the macro. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. The execute form of the macro stores the parameters into the storage area defined by mfctrl and provides full syntax checking with the default setting.

\texttt{\textasciitilde mfctrl}

This is a required keyword that specifies a storage area for the parameter list. This can be an RS-type address or an address in register (1)-(12).

\texttt{\textasciitilde mfattr}

An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code \texttt{mfattr}, the system provides a value of 0D.

\texttt{\textasciitilde COMPLETE}

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

None.

Return codes

When the IRDFSDU macro returns control to your program:

- GPR 15 (and retcode, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSN CODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes:

\textit{Table 96. Return Codes for IRDFSDU macro}

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td>Meaning: Successful completion. Data updated for all FICON switches.</td>
</tr>
<tr>
<td>04</td>
<td>2</td>
<td>Warning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: The server task is not available</td>
</tr>
<tr>
<td>04</td>
<td>24</td>
<td>Environmental Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: FICON switch statistics disabled</td>
</tr>
<tr>
<td>08</td>
<td>1</td>
<td>Meaning: The subpool provided by the caller is not in common storage.</td>
</tr>
<tr>
<td>08</td>
<td>2</td>
<td>Meaning: The switch device provided by the caller is not in the Switch Table.</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>Meaning: An ESTAE could not be established.</td>
</tr>
</tbody>
</table>
## Reason codes

The following table identifies the hexadecimal return and reason codes:

**Table 97. Return and Reason Codes for IRDFSD macro**

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>04</td>
<td>Required module was not found. The request could not be processed.</td>
</tr>
<tr>
<td>00000002</td>
<td>04</td>
<td>The server task is unavailable. The request could not be processed.</td>
</tr>
<tr>
<td>00000024</td>
<td>04</td>
<td>FICON switch statistics disabled.</td>
</tr>
<tr>
<td>00000003</td>
<td>08</td>
<td>Caller is not APF authorized.</td>
</tr>
<tr>
<td>00000008</td>
<td>08</td>
<td>IRDVFS does not recognize the request type.</td>
</tr>
<tr>
<td>00000023</td>
<td>08</td>
<td>IOSVFS does not recognize the request type.</td>
</tr>
<tr>
<td>00000004</td>
<td>0C</td>
<td>Recovery could not be established.</td>
</tr>
<tr>
<td>00000005</td>
<td>0C</td>
<td>POST failed.</td>
</tr>
<tr>
<td>00000006</td>
<td>0C</td>
<td>Recovery was entered.</td>
</tr>
<tr>
<td>00000025</td>
<td>0C</td>
<td>1/O error.</td>
</tr>
<tr>
<td>00000026</td>
<td>0C</td>
<td>UCBSCAN failure.</td>
</tr>
<tr>
<td>00000027</td>
<td>0C</td>
<td>UCBLOOK failure.</td>
</tr>
<tr>
<td>00000028</td>
<td>0C</td>
<td>Server task recovery entered.</td>
</tr>
</tbody>
</table>
IRDFS DU macro
Chapter 112. ISGADMIN — Global resource serialization administration service

Description

Interface for Global Resource Serialization Administration

The GRS Administration service routine is given control from the ISGADMIN macro to:

- Change maximum ENQ limits for a specific address space.
- Move an ENQ waiter to a different position in the request queue and to optionally change its control type from exclusive to shared.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>The caller must be authorized, although any one of the following attributes is sufficient:</td>
</tr>
<tr>
<td></td>
<td>Supervisor State</td>
</tr>
<tr>
<td></td>
<td>Key 0-7</td>
</tr>
<tr>
<td></td>
<td>APF-authorized</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td></td>
<td>Note: The updated ENQ limit is updated for the home address space.</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td></td>
<td>If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td></td>
<td>If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>The caller must not be locked.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller’s dispatchable unit access list (DU-AL).</td>
</tr>
<tr>
<td></td>
<td>The control parameters must be in the same key as the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

The caller must include the ISGYCON macro to get the return and reason codes.

Restrictions

The caller must not have functional recovery routines (FRRs)

This macro supports multiple versions. Some keywords are unique to certain versions. See the PLISTVER parameter description.
### Input register information
Before issuing the ISGADMIN macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

### Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR15 is not 0</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

### Performance implications
None.
Syntax

Note:

main diagram

REQUEST=SETENQMAX, MAXTYPE=AUTHORIZED, MAXVALUE=maxvalue
REQUEST=RESETENQMAX, MAXTYPE=AUTHORIZED
REQUEST=RESETENQMAX, MAXTYPE=AUTHORIZED
REQUEST=MOVEWAITER, parameters-1

Note:

parameters-1

MOVINGWAITER=movingwaiter, NEWCONTROL=EXCLUSIVE

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the ISGADMIN macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,BEFOREREQUESTER=beforerequester
When TOTHEEND=NO and REQUEST=MOVEWAITER are specified, a required input parameter that is an ENQToken identifying the ENQ request that the MovingWaiter request precedes.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,MAXTYPE=AUTHORIZED
,MAXTYPE=AUTHORIZED
When REQUEST=SETENQMAX is specified, a required parameter.
ISGADMIN macro

\texttt{\texttt{MAXTYPE=AUTHORIZED}}
\begin{itemize}
\item Indicates a request to update the maximum ENQ limit for authorized requesters.
\end{itemize}

\texttt{\texttt{MAXTYPE=UNAUTHORIZED}}
\begin{itemize}
\item Indicates a request to update the maximum ENQ limit for unauthorized requesters.
\end{itemize}

\texttt{\texttt{MAXTYPE=AUTHORIZED}}
\texttt{\texttt{MAXTYPE=UNAUTHORIZED}}
\begin{itemize}
\item When REQUEST=RESETENQMAX is specified, a required parameter.
\end{itemize}

\texttt{\texttt{MAXTYPE=AUTHORIZED}}
\texttt{\texttt{MAXTYPE=UNAUTHORIZED}}
\begin{itemize}
\item Indicates a request to reset the maximum ENQ limit for authorized requesters.
\item Indicates a request to reset the maximum ENQ limit for unauthorized requesters.
\end{itemize}

\texttt{\texttt{MAXVALUE=maxvalue}}
\begin{itemize}
\item When REQUEST=SETENQMAX is specified, a required input parameter that is the requested value of the new maximum ENQ limit. The specified value must be greater than or equal to the absolute minimum described in ISGYCON, and up to $2^{31}-1$ (2147483647).
\end{itemize}

\texttt{\texttt{To code:}} Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

\texttt{\texttt{MF=S}}
\texttt{\texttt{MF=(L, list addr)}}
\texttt{\texttt{MF=(L, list addr, attr)}}
\texttt{\texttt{MF=(L, list addr, 0D)}}
\texttt{\texttt{MF=(E, list addr)}}
\texttt{\texttt{MF=(E, list addr, COMPLETE)}}
\begin{itemize}
\item An optional input parameter that specifies the macro form.
\end{itemize}

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

\texttt{\texttt{list addr}}
\begin{itemize}
\item The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).
\end{itemize}

\texttt{\texttt{attr}}
\begin{itemize}
\item An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

When REQUEST=MOVEWAITER is specified, a required input parameter that is an ENQToken identifying the ENQ waiter.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

When REQUEST=MOVEWAITER is specified, an optional parameter. The default is NEWCONTROL=EXCLUSIVE.

Indicates that the requester represented by the MovingWaiter ENQToken should have its control remain Exclusive.

Indicates that the request represented by the MovingWaiter ENQToken should have its control become Shared.

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- IMPLIED_VERSION, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- MAX, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
  
  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- 1, which supports all parameters except those specifically referenced in higher versions.
- 2, which supports both the following parameters and those from version 1:
  - BEFOREREQUESTER
  - MOVINGWAITER
  - NEWCONTROL
  - TOTHEEND

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
ISGADMIN macro

- A decimal value of 1, or 2

REQUEST=SETENQMAX
REQUEST=RESETENQMAX
REQUEST=MOVEWAITER

A required parameter that indicates the type of ISGADMIN request.

REQUEST=SETENQMAX
Indicates a request to change the ENQ maximum for the home address space.

REQUEST=RESETENQMAX
Indicates a request to reset the ENQ maximum for the home address space back to the system default.

REQUEST=MOVEWAITER
Indicates a request to move an ENQ waiter to a different position in the request queue and to optionally change its control type through the NEWCONTROL keyword.

This request requires a version 2 parameter list.

Note: This function is intended to only be used by third party serialization products. Its misuse can result in deadlocks, incorrect serialization or loss of data integrity. The MOVEWAITER, TOTHEEND, and BEFOREREQUESTER keywords specify which requester should be moved and where to move it. The waiter will only be moved under the following conditions:

- The MOVINGWAITER:
  - Has a requested disposition of Exclusive
  - Is not currently an owner of the resource
  - Cannot result in any new owners as a result of the move
  - Must be waiting for the same resource as the BEFOREREQUESTER (if specified)
- The resource is NOT global or STEP in scope. Note that in GRS=NONE mode, the final scope can be SYSTEMS or SYSTEM. When in other GRS modes the scope must be SYSTEM.

,RETCODE=retcode
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=rsncode
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0, (REG00), or (R0).

,TOTHEEND=NO
,TOTHEEND=YES

When REQUEST=MOVEWAITER is specified, an optional parameter. The default is TOTHEEND=NO.
Indicates that the requester represented by the MovingWaiter ENQToken should be moved to a position specified through the BEFOREREQUESTER keyword.

Indicates that the request represented by the MovingWaiter ENQToken should be moved to the end of the request queue.

**ABEND codes**

None

**Return and reason codes**

When the ISGADMIN macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro ISGYCON provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the xxxx value.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>—</td>
<td>Equate Symbol: ISGADMINRc_OK</td>
<td>ISGADMIN request successful.</td>
<td>None required.</td>
</tr>
<tr>
<td>04</td>
<td>—</td>
<td>Equate Symbol: ISGADMINRc_Warn</td>
<td>Warning</td>
<td>Refer to action under the individual reason code.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0401</td>
<td>Equate Symbol: ISGADMINRsn_ENQMaxValueLow</td>
<td>For REQUEST=SETENQMAX. The specified maximum is less than or equal to the current system-wide maximum. This space-specific maximum has been set but has no immediate effect.</td>
<td>Ensure the specified MaxValue is accurate. If not, reissue the ISGADMIN service with a higher value.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0402</td>
<td>Equate Symbol: ISGADMINRsn_ResetENQMaxIgnored</td>
<td>For REQUEST=RESETENQMAX. The home address space did not have a specific maximum for that type of requester.</td>
<td>Ensure that the reset was desired, and issued for the appropriate requester type, authorized or unauthorized. Reissue the service with the correct requester type if appropriate.</td>
</tr>
<tr>
<td>08</td>
<td>—</td>
<td>Equate Symbol: ISGADMINRc_ParmError</td>
<td>ISGADMIN request specified parameters in error.</td>
<td>Refer to action under the individual reason code.</td>
</tr>
</tbody>
</table>
### ISGADMIN macro

**Table 98. Return and Reason Codes for the ISGADMIN Macro (continued)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>xxxx0801</td>
<td><strong>Equate Symbol</strong>: ISGADMINRsn_BadPlstAddress</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Unable to access parameter list.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check that the parameter list is addressable. If in AR-mode, check that the ALET of the parameter list is correct. Note that if this macro is issued in AR-mode, SYSSTATE ASCENV=AR must be issued before this macro. Ensure that the storage is in the same key as the caller.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx0802</td>
<td><strong>Equate Symbol</strong>: ISGADMINRsn_BadPlstALET</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Bad parameter list ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller’s Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Make sure that the ALET of the parameter list is valid. Its access register may not have been set up properly.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx0803</td>
<td><strong>Equate Symbol</strong>: ISGADMINRsn_BadPlstVersion</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Bad parameter list version number. The ISGADMIN parameter list version is greater than the version supported by GRS on the current system or the ISGADMIN parameter list version is lower than the minimum required for parameters that were specified.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check for possible storage overlay of the parameter list. Retry the request with the correct version number. Verify that your program was assembled with the correct macro library for the release of MVS on which your program is running.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx0804</td>
<td><strong>Equate Symbol</strong>: ISGADMINRsn_ReservedFieldNotNull</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: A reserved field in the parameter list is non-zero.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check for possible storage overlay of the parameter list.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx0805</td>
<td><strong>Equate Symbol</strong>: ISGADMINRsn_BadRequest</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Bad REQUEST parameter.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: IBM suggests that the ISGADMIN macro is used when invoking the ISGADMIN service.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx0806</td>
<td><strong>Equate Symbol</strong>: ISGADMINRsn_ENQMaxValueTooLow</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: For REQUEST=SETENQMAX. The specified maximum is less than the smallest allowable maximum.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check the smallest allowable maximum in macro ISGYCON. Reissue the ISGADMIN service with a higher value.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx0807</td>
<td><strong>Equate Symbol</strong>: ISGADMINRsn_BadMovingWaiterAddress</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: For REQUEST=MOVEWAITER. Unable to access the MovingWaiter ENQToken.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Make sure that the MovingWaiter ENQToken is addressable. If in AR-mode, this field is accessed through its address and ALET, check that both these values are correct. Ensure that the storage is in the same key as the caller.</td>
<td></td>
</tr>
</tbody>
</table>
Table 98. Return and Reason Codes for the ISGADMIN Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0808    | **Equate Symbol**: ISGADMINRsn_BadMovingWaiterAlet  
**Meaning**: For REQUEST=MOVEWAITER. Bad MovingWaiter ENQToken ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.  
**Action**: Make sure that the ALET of the MovingWaiter ENQToken is valid. Its access register may not have been set up properly. |
| 08          | xxxx0809    | **Equate Symbol**: ISGADMINRsn_BadMovingWaiter  
**Meaning**: For REQUEST=MOVEWAITER. The specified MovingWaiter ENQToken does not represent an ENQ on the current system.  
**Action**: Make sure that the specified MovingWaiter ENQToken is from a previous request that has not been subsequently released. |
| 08          | xxxx080A    | **Equate Symbol**: ISGADMINRsn_BadBeforeRequesterAddress  
**Meaning**: For REQUEST=MOVEWAITER. Unable to access the BeforeRequester ENQToken.  
**Action**: Make sure that the BeforeRequester ENQToken is addressable. If in AR-mode, this field is accessed through its address and ALET, check that both these values are correct. Ensure that the storage is in the same key as the caller. |
| 08          | xxxx080B    | **Equate Symbol**: ISGADMINRsn_BadBeforeRequesterAlet  
**Meaning**: For REQUEST=MOVEWAITER. Bad BeforeRequester ENQToken The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.  
**Action**: Make sure that the ALET of the BeforeRequester ENQToken is valid. Its access register may not have been set up properly. |
| 08          | xxxx080C    | **Equate Symbol**: ISGADMINRsn_BadBeforeRequester  
**Meaning**: For REQUEST=MOVEWAITER. The specified BeforeRequester ENQToken does not represent an ENQ on the current system.  
**Action**: Make sure that the specified BeforeRequester ENQToken is from a previous request that has not been subsequently released. |
| 08          | xxxx080D    | **Equate Symbol**: ISGADMINRsn_SameRequester  
**Meaning**: For REQUEST=MOVEWAITER. The same ENQToken was specified for both MovingWaiter and BeforeRequester.  
**Action**: Make sure that the ENQTokens are distinct. |
### ISGADMIN macro

**Table 98. Return and Reason Codes for the ISGADMIN Macro (continued)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx080E      | **Equate Symbol**: ISGADMINRsn_InconsistentResource  
**Meaning**: For REQUEST=MOVEWAITER. The specified MovingWaiter and BeforeRequester ENQTokens do not represent ENQ requests for the same resource.  
**Action**: Make sure that the ENQTokens specified are against the same resource. |
| 08          | xxxx080F      | **Equate Symbol**: ISGADMINRsn_BadScope  
**Meaning**: For REQUEST=MOVEWAITER. The resource associated with the specified MovingWaiter and BeforeRequester ENQTokens is not a valid local resource. The resource cannot be a global or a STEP resource. Note that in GRS=NONE mode, an acceptable local resource can have a final scope of SYSTEMS or SYSTEM. When in other GRS modes, the final scope can only be SYSTEM.  
**Action**: Make sure that the ENQTokens specified are against a valid local resource. |
| 08          | xxxx0810      | **Equate Symbol**: ISGADMINRsn_BadControl  
**Meaning**: For REQUEST=MOVEWAITER. The specified MovingWaiter ENQToken represents a requester of shared control.  
**Action**: Make sure that the MovingWaiter ENQToken represents a requester of exclusive control. |
| 08          | xxxx0811      | **Equate Symbol**: ISGADMINRsn_CannotMoveOwner  
**Meaning**: For REQUEST=MOVEWAITER. If granted, the requester represented by the MovingWaiter ENQToken would have become the owner of the resource because it would precede an owner.  
**Action**: Make sure that the BeforeRequester ENQToken represents a waiting requester. |
| 08          | xxxx0812      | **Equate Symbol**: ISGADMINRsn_AlreadyBeforeRequester  
**Meaning**: For REQUEST=MOVEWAITER, TOTHEEND=NO, BEFOREREQUESTER=. The specified MovingWaiter ENQToken represents a requester that is already queued just before the requester represented by the BeforeRequester ENQToken. The control was not changed.  
**Action**: Make sure that the MovingWaiter and BeforeRequester ENQTokens represent the correct requesters and that the queue is as expected. |
| 08          | xxxx0813      | **Equate Symbol**: ISGADMINRsn_CannotMoveBeforeOwner  
**Meaning**: For REQUEST=MOVEWAITER. If granted, the requester represented by the MovingWaiter ENQToken would have become the owner of the resource because it would precede an owner.  
**Action**: Make sure that the BeforeRequester ENQToken represents a waiting requester. |
### Table 98. Return and Reason Codes for the ISGADMIN Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>xxxx0814</td>
<td><strong>Equate Symbol:</strong> ISGADMINRsn_CannotMoveAfterSharedOwner</td>
<td>For REQUEST=MOVEWAITER, NEWCONTROL=SHARED. If granted, the requester represented by the MovingWaiter ENQToken would have become the owner of the resource because it would immediately follow a shared owner.</td>
<td>Action: Make sure that the BeforeRequester ENQToken represents the requester that the moving waiter should precede.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0815</td>
<td><strong>Equate Symbol:</strong> ISGADMINRsn_CannotMakeAnotherOwner</td>
<td>For REQUEST=MOVEWAITER. If granted, one or more requesters queued after the one represented by the MovingWaiter ENQToken would have become the owner of the resource.</td>
<td>Action: Make sure that the MOVEWAITER request would not make any other waiting requesters the owner of the resource.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0816</td>
<td><strong>Equate Symbol:</strong> ISGADMINRsn_AlreadyLastRequester</td>
<td>For REQUEST=MOVEWAITER, TOTHEEND=YES. The requester represented by the MovingWaiter ENQToken is already at the end of the request queue.</td>
<td>Action: Make sure that the MovingWaiter ENQToken represents a requester at the correct position and that the request queue is as expected.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0817</td>
<td><strong>Equate Symbol:</strong> ISGADMINRsn_CannotMoveMasidUser</td>
<td>For REQUEST=MOVEWAITER. The MovingWaiter ENQToken represents a MASID user.</td>
<td>Action: Make sure that the MovingWaiter ENQToken does not represent a MASID user.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0818</td>
<td><strong>Equate Symbol:</strong> ISGADMINRsn_MasidControlConflict</td>
<td>For REQUEST=MOVEWAITER, NEWCONTROL=SHARED. The requester represented by the MovingWaiter ENQToken would create a bad MASID environment since a shared owner of the resource is a convert-to-exclusive MASID target.</td>
<td>Action: Make sure that the requester represented by the MovingWaiter ENQToken would not need to move in the midst of a MASID convert-to-exclusive environment or that the moved requester could maintain a control of Exclusive.</td>
</tr>
<tr>
<td>0C</td>
<td></td>
<td><strong>Equate Symbol:</strong> ISGADMINRc_EnvError</td>
<td>ISGADMIN request has an environment error.</td>
<td>Refer to action under the individual reason code.</td>
</tr>
<tr>
<td>0C</td>
<td>xxxx0C01</td>
<td><strong>Equate Symbol:</strong> ISGADMINRsn_NotAuthorized</td>
<td>An unauthorized caller invoked the ISGADMIN service.</td>
<td>An ISGADMIN caller must be authorized.</td>
</tr>
</tbody>
</table>
Table 98. Return and Reason Codes for the ISGADMIN Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
</table>
| 0C          | xxxx0C02    | Equate Symbol: ISGADMINRsnn_FRRHeld  
Meaning: The caller issued ISGADMIN when an FRR was established.  
Action: Avoid issuing ISGADMIN when using functional recovery routines. |
| 0C          | xxxx0C03    | Equate Symbol: ISGADMINRsnn_LockHeld  
Meaning: A lock was held upon entry. No locks may be held when calling ISGADMIN.  
Action: Avoid using ISGADMIN when locks are held. |
| 0C          | xxxx0C04    | Equate Symbol: ISGADMINRsnn_SrbMode  
Meaning: SRB mode.  
Action: SRB mode is not supported. |
| 0C          | xxxx0C05    | Equate Symbol: ISGADMINRsnn_NotEnabled  
Meaning: Not Enabled.  
Action: Avoid using ISGADMIN when not enabled. |
| 0C          | xxxx0C06    | Equate Symbol: ISGADMINRsnn_QueueDamage1  
Meaning: The GRS resource queue structure for the target resource is damaged. Further processing against the queue is not allowed.  
Action: Prevent any further processing against the target resource. |
| 0C          | xxxx0C07    | Equate Symbol: ISGADMINRsnn_QueueDamage2  
Meaning: The GRS resource queue structure for the target resource is damaged. Further processing against the queue is not allowed.  
Action: Prevent any further processing against the target resource. |
| 10          | —           | Equate Symbol: ISGADMINRc_CompError  
Meaning: Component Error  
Action: Contact the IBM Support Center. Provide the reason code which contains diagnostic data. |

Examples

* ******************************************************************************
* Set the unauthorized ENQ maximum for the home address space
* ******************************************************************************

```
ISGADMIN REQUEST=SETENQMAX,  
MAXTYPE=UNAUTHORIZED,MAXVALUE=MYVALUE,  
RETCODE=MYRC,RSNCODE=MYRSN
```

* ******************************************************************************
* Reset the unauthorized ENQ maximum of the home address space
* ******************************************************************************
**ISGADMIN macro**

```
ISGADMIN REQUEST=RESETENQMAX,
   MAXTYPE=UNAUTHORIZED,
   RETCODE=MYRC,RSCODE=MYRSN

* Move an ENQ Waiter
* ******************************************************

ISGADMIN REQUEST=MOVEWAITER,
   MOVINGWAITER=mywaiterENQToken,
   TOTHEEND=NO,
   BEFOREREQUESTER=mybeforerequesterENQToken,
   RETCODE=MYRC,RSCODE=MYRSN
```

For more information about global resource serialization, see z/OS MVS Planning: Global Resource Serialization.
Chapter 113. ISGEC — GRS enhanced contention analysis service

Description

Use the ISGEC service to obtain waiter and blocker information for global resource serialization (GRS) component managed resources. GRS resource waiter/blocker information can be obtained for a specific system within the current sysplex, or for all the systems operating in the current sysplex.

A GRS resource is considered relevant to an ISGEC request if that resource currently has waiters and blockers associated with it. For a given relevant GRS resource, ISGEC returns the following types of information:

**Waiter**
The longest waiting unit of work for that resource, and the top (longest) blocking unit of work for that waiter. Further general information about the resource and the numbers of resource owners and waiters is also reported.

**Blocker**
The longest blocking unit of work for that resource. Further general information about the resource and the numbers of resource owners and waiters is also reported.

ISGEC returns information for as many relevant GRS resources as is specified by the COUNT parameter. All reported resource information is collected into a virtual storage buffer specified by the RIBOUT parameter. Reported information is formatted according to RIB and RIBE DSECTs, available from syslib member ISGRIB. See WAITER and BLOCKER descriptions under the "REQUEST=WAITER" on page 1126 parameter for the specific RIBOUT buffer area format. For precise descriptions of resource, waiter and blocker information reported, see "RIB Heading Information" in z/OS MVS Data Areas in the Z/OS Internet library [http://www.ibm.com/systems/z/os/zos/bkser/](http://www.ibm.com/systems/z/os/zos/bkser/).

ISGEC reports on relevant resources as they are encountered in the system’s GRS resource management data infrastructure. The order of reported resources in the RIBOUT area is unpredictable, and implies no suggestion of one resource having greater waiter/blocker considerations than any other reported on resource.

The ISGEC service might be unable to report any waiter or blocker information for some sysplex systems, in some invocation cases, for a variety of reasons. In the event that this occurs, ISGEC reports the system names of systems not included in the report, and the reason for not including those systems, in the NOTINCL output area. The description for parameter NOTINCL explains the output area format and reason codes associated with it.

**Note:** The 476-byte (or X'1DC') parameter list constructed by ISGEC and passed to its service routine MUST reside in common area subpool 231. This requirement has significant implications on the use of the various macro format (MF) options. For more information about this parameter list requirement, see "Programming requirements" on page 1122.
### Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state. Zero PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary, Secondary, access register (AR), or Home</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks may be held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space.</td>
</tr>
</tbody>
</table>

### Programming requirements

1. The parameter list constructed by ISGECA and passed to its service routine MUST reside in common area subpool 231. This has the following implications:
   - For assembler standard format invocations (i.e., MF=(S)), the invoking program code must reside in subpool 231, as an inline parameter list is generated.
   - For PL/X standard format invocations (i.e., MF(S)), the invoker's dynamic area must reside in subpool 231, as a dynamic area declare for the parameter list is generated.
   - Similarly, for list format invocations (i.e., MF=(L,xxx)), if the resulting declared parameter list resides in the program's dynamic storage area, then this storage must be obtained from subpool 231.
     - If the resulting list format parameter list declare is a PL/X based construct, then the program may substantiate the based construct via an allocated subpool 231 address for subsequent execute format (i.e., MF=(E,xxx)) invocations.
   - For execute format invocations (i.e., MF=(E,xxx)), the specified parameter list must reside in common area subpool 231.
     - The parameter list must be 476 (or X'1DC') bytes in length.
2. PL/X invokers must include syslib members CVT and ISGGVT.
3. Include syslib member ISGRIB for RIB and RIBE DSECT mappings. These DSECTs precisely describe formatted areas in the RIBOUT area.
4. ISGECAs service return and reason codes can be retrieved from the ISGECAs parameter list area, as an alternative to coding the RETCODE and RSNCODE parameters. These results appear in the parameter list as follows:
   - Return code: 2-byte value at offset 60 (or X'3C').
   - Reason code: 2-byte value at offset 62 (or X'3E').
5. The ISGECAs service requires a specific system service or release level to function successfully. The ISGECAs macro expansion performs before any other tests and calling the service routine, verifying the system has this function enabled.

### Restrictions

None
ISGECA macro

Input register information
Before issuing the ISGECA macro, the caller does not have to place any
information into any register unless using it in register notation for a particular
parameter, or using it as a base register.

Before issuing the ISGECA macro, the caller does not have to place any
information into any access register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unchanged</td>
</tr>
<tr>
<td>1</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>15</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unchanged</td>
</tr>
<tr>
<td>1</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>15</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>

Performance implications
None.
ISGEC macro

Syntax

Note:

main diagram

```
\begin{center}
\begin{tikzpicture}
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (name) at (0,0) {name};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (ISGECA) at (1.5,0) {ISGECA};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (REQUEST) at (3,0) {REQUEST};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (WAITER) at (4.5,0) {WAITER};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (REQUEST) at (6,0) {REQUEST};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (BLOCKER) at (7.5,0) {BLOCKER};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (COUNT) at (0,-1.5) \text{COUNT} = \text{count};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (SYSTEM) at (1.5,-1.5) \text{SYSTEM}\text{,} \text{SYSNAME} = \text{sysname};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (COUNT) at (3,-1.5) \text{COUNT} = \text{count};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (RIBOUT) at (4.5,-1.5) \text{RIBOUT}\text{,} \text{RIBOUTLN} = \text{riboutln};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (RIBOUTCT) at (6,-1.5) \text{RIBOUTCT} = \text{riboutct};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (NOTINCL) at (0,-3) \text{NOTINCL} = \text{notincl};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (NOTINCCT) at (1.5,-3) \text{NOTINCCT} = \text{notincct};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (RSNCODE) at (3,-3) \text{RSNCODE} = \text{rsncode};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (plistver) at (4.5,-3) \text{PLISTVER} = \text{IMPLIES VERSION};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (list addr) at (0,-4.5) \text{(L, list addr)};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (list addr) at (1.5,-4.5) \text{(L, list addr, attr)};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (list addr) at (3,-4.5) \text{(L, list addr, OD)};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (list addr) at (4.5,-4.5) \text{(L, list addr, COMPLETE)};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (list addr) at (6,-4.5) \text{(E, list addr)};
  \node[draw,rectangle,minimum width=1.5cm,minimum height=1cm] (list addr) at (7.5,-4.5) \text{(E, list addr, COMPLETE)};

\end{tikzpicture}
\end{center}
```

Parameters

The parameters are explained as follows:

\textit{name}

An optional symbol, starting in column 1, that is the name on the ISGEC macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

\textit{COUNT=count}

A required input parameter describing the maximum number of relevant resources to be reported on by this ISGEC invocation. The maximum value that can be specified with the COUNT parameter is 99.

To code: Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

\textit{MF=S}
\textit{MF=(L, list addr)}
\textit{MF=(L, list addr, attr)}
\textit{MF=(L, list addr, OD)}
\textit{MF=(E, list addr)}
\textit{MF=(E, list addr, COMPLETE)}

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.
ISGEC A macro

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the P LISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
   The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
   An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

,COMPLETE
   Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOTINCC T=notincct
   An optional output parameter, to contain the number of systems reported in the NOTINCL area. Alternatively, this number of NOTINCL entries can be obtained from the 2-byte parameter list field at offset 58 (or X'3A').

   To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,NOTINCL=notincl
   A required input parameter that contains the address of a virtual storage output area to contain the list of systems for which RIBs and RIBEs are not included in the RIBOUT area. The NOTINCL area must begin on a doubleword boundary, and must reside within common storage subpool 231.

   The length of the NOTINCL area, in bytes, must, minimally, be the number of systems currently executing in the sysplex multiplied by 10 (or X'0A'). The format of the NOTINCL area is as follows:

   +---------------------------+
   | System name | Reason Code |
   | System name | Reason Code |
   | System name | Reason Code |
   +---------------------------+

   Each system name and reason code pair potentially reflects a system not included in waiter/blocker data returned in the RIBOUT area. The number of systems reported on in the NOTINCL area is returned in the NOTINCC T output parameter value.

   Each NOTINCL system name field is an 8-byte field, and each reason code entry is a 2-byte field. Reason codes for the NOTINCL area are independent of ISGEC A service invocation reason codes, and are only meaningful when the ISGEC A return code is 4 or less. The NOTINCL reason codes and meanings are as follows:

   Hex Reason Code
   Meaning
ISGEC macro

0000  Ignore this NOTINCL area entry, including the system name value specified.

0001  The system described by the system name field is cannot process the ISGEC service.

0002  The system described by the system name field was not found to be participating in the current sysplex.

0003  The system described by the system name field did not respond to an XCF request to gather ISGEC report information.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,,PLISTVER=IMPLIED_VERSION
,,PLISTVER=MAX
,,PLISTVER=0

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- IMPLIED_VERSION, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- MAX, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- 0, if you use the currently available parameters.

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0

REQUEST=WAITER
REQUEST=BLOCKER

A required parameter that indicates the type of ISGEC request.

REQUEST=WAITER

When you specify WAITER, the longest waiters and top blockers for each relevant resource are returned. For REQUEST=WAITER, the RIBOUT buffer area is formatted as follows:

```
+-------------------+
| RIB | RIBE | RIBE |
| RIB | RIBE | RIBE |
| RIB | RIBE | RIBE |
+-------------------+
```

The number of RIBs collected in the RIBOUT area is returned in the RIBOUTCT parameter variable. Each RIB/RIBE/RIBE trio reports on the following:
The RIB describes general information about the resource, including the QNAME, minor name and the numbers of waiters and blockers.

The first RIBE describes the top blocking unit of work for this resource.

The second RIBE describes the longest waiting unit of work for this resource.

REQUEST=BLOCKER

When you specify BLOCKER, the top blockers for each relevant resource is returned. For REQUEST=BLOCKER, the RIBOUT buffer area is formatted as follows:

```
+------------+
| RIB | RIBE |
| RIB | RIBE |
| RIB | RIBE |
+------------+
```

The number of RIBs collected in the RIBOUT area is returned in the RIBOUTCT parameter variable. Each RIB/RIBE pair reports on the following items:

- The RIB describes general information about the resource, including the QNAME, minor name and the numbers of waiters and blockers.
- The RIBE describes the top blocking unit of work for this resource.

**,RETCODE=retcode**

An optional output parameter that will contain the return code.

**To code:** Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,**RIBOUT=ribout**

A required input parameter that contains the address of the virtual storage output area for this request. The RIBOUT area must reside in the invoker's primary address space, and contains the ISGECA report of RIBs and RIBEs for the request.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,**RIBOUTCT=riboutct**

An optional output parameter, to contain the number of RIBs collected in the RIBOUT area. Alternatively, this number of RIBs can be obtained from the 2-byte parameter list field at offset 56 (or X'38').

**To code:** Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,**RIBOUTLN=riboutln**

A required input parameter describing the length, in bytes, of the RIBOUT virtual storage area.

The length of the RIBOUT area must be large enough to accommodate the maximum size ISGECA report for the request, and therefore must be of a magnitude that facilitates the COUNT parameter value and RIB/RIBE DSECT mapping sizes. Depending on the ISGECA request type, this relationship between these parameter values and DSECT sizes can be expressed as follows:

**Waiter:**

RIBOUTLN parameter value must equal or exceed the COUNT parameter value multiplied by 392 (or X'188').
ISGEC macro

Blocker:

RIBOUTLN parameter value must equal or exceed the COUNT parameter value multiplied by 344 (or X'158').

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,RSNCODE=rsncode

An optional output parameter that will contain the reason code.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), (REG0), (REG00), or (R0).

,SCOPE=SYSTEM

A required parameter that indicates the request scope.

,SCOPE=SYSTEM

ISGEC is to only report on blockers and, potentially, waiters currently executing on a specific system within the current GRS complex.

,SCOPE=SYSTEMS

ISGEC is to report on blockers and, potentially, waiters across all of the systems in the current sysplex complex.

,SYSNAME=sysname

When SCOPE=SYSTEM is specified, a required input parameter string containing the system name of the single system on which ISGEC is to report.

SYSNAME is required when you specify SCOPE=SYSTEM. SYSNAME is not valid for SCOPE=SYSTEMS.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

ABEND codes

None.

Return and reason codes

When the ISGEC macro returns control to your program:

- GPR 15 (and retcode, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes. IBM support personnel may request the entire reason code, including the xxxx value.

Table 99. Return and Reason Codes for the ISGEC Macro

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>–</td>
<td>Successful completion. The RIBOUT virtual storage area contains the waiter or blocker output report, and the NOTINCL virtual storage area describes system names and reason codes for systems not reported on in the RIBOUT area.</td>
</tr>
<tr>
<td>04</td>
<td>–</td>
<td>Request completed with exceptional circumstances.</td>
</tr>
</tbody>
</table>
### Table 99. Return and Reason Codes for the ISGECA Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04          | xxxx0000    | **Meaning:** The ISGECA service could not communicate with some systems in the sysplex, so the returned data is incomplete. This implies that such systems are non-responsive, such as when a system has been reset but has not been removed from the sysplex.  
**Action:** No suggested program action. |
| 04          | xxxx0001    | **Meaning:** Not all systems in the current GRS complex that are relevant to the ISGECA request are participating in the current sysplex.  
**Action:** No suggested program action. The RIBOUT virtual storage area contains the waiter or blocker output report, and the NOTINCL virtual storage area describes system names and reason codes for systems not reported on in the RIBOUT area. |
| 08          | –           | Request failed. |
| 08          | xxxx0000    | **Meaning:** A GRS internal error occurred and the request could not be completed.  
**Action:** No suggested program action. |
| 08          | xxxx0001    | **Meaning:** The ISGECA service routine was unable to obtain storage necessary to process the request.  
**Action:** Consider reducing the COUNT and RIBOUTLN parameter values to decrease the total number of resources to be reported, and re-invoke ISGECA. Alternatively, if this was a SCOPE(SYSTEMS) request, consider re-invoking ISGECA with SCOPE(SYSTEM) to, potentially, reduce the total number of resources to be reported. |
| 08          | xxxx0003    | **Meaning:** A GRS internal error occurred and the request could not be completed.  
**Action:** No suggested program action. |
| 08          | xxxx0004    | **Meaning:** The sysplex is in the process of migrating to GRS STAR mode, and therefore cannot process the request at this time.  
**Action:** Iteratively retry the ISGECA invocation, waiting a few seconds between attempts. |
| 08          | xxxx00FD    | **Meaning:** The maximum number of relevant resources to be reported on, as specified by parameter COUNT, exceeds the service maximum value of 99.  
**Action:** Correct the COUNT parameter value and reinvoke ISGECA. |
| 08          | xxxx00FE    | **Meaning:** The RIBOUT length specified in parameter RIBOUTLN was not large enough to process the number of resource requests specified by parameter COUNT.  
**Action:** Correct the RIBOUTLN or COUNT parameter values and reinvoke ISGECA. |
| 08          | xxxx00FF    | **Meaning:** ISGECA is an unsupported service on this system.  
**Action:** No suggested program action. The system needs a service or release level upgrade before ISGECA can be successfully invoked. |
Examples

The following examples do not show, but presume, the existence of appropriate assembler continuation characters in column 72. The examples also presume an appropriate assembler storage declaration for each instance of a named symbol ISGECAP parameter.

The first example depicts an invocation of ISGECAP to collect waiter data for a specific sysplex system, whose 8-character system name is stored at program location MYSYSNAME:

```
XR 2,2 Clear reg 2
LHI 2,476 ISGECAP parm list length into R2
STORAGE OBTAIN,LENGTH=(2),ADDR=(3),SP=231,COND=NO
GETWAIT ISGECAP REQUEST=WAITER,SCOPE=SYSTEM,
SYSNAME=MYSYSNAME,RIBOUT=OUTAREA@,
RIBOUTLN=MYSYSNAME,RIBOUTCT=MYRIBCT,
COUNT=MYCOUNT,RIBINCL=NOTINCLAREA@,
RIBINCT=MYRIBCT,RETCODE=MYRETCODE,
RSNMCODE=MYRSCNMCODE,PLISTVER=MAX,
MF=(E,(3))
```

For the above, subpool 231 storage is obtained and then passed through the MF= parameter for the ISGECAP service routine parameter list.

Upon return from the service routine, the virtual storage area specified by OUTAREA@ contains the waiter report RIBs and RIBEs for up to MYCOUNT number of resources; while the virtual storage area specified by NOTINCLAREA@ contains the associated list of systems (with reasons) that are not included in the RIBOUT area report. The precise number of RIBs returned in the OUTAREA@ area is returned in the MYRIBCT program variable.

This second example depicts an invocation of ISGECAP to collect blocker data for all the systems in the current sysplex:

```
GETBLOCK ISGECAP REQUEST=BLOCKER,SCOPE=SYSTEMS,
RIBOUT=OUTAREA@,RIBOUTLN=MYSYSNAME,RIBOUTCT=MYRIBCT,
COUNT=MYCOUNT,RIBINCL=NOTINCLAREA@,
RIBINCT=MYRIBCT,RETCODE=MYRETCODE,
RSNMCODE=MYRSCNMCODE,PLISTVER=MAX, MF=S
```

Parameter usage and results for this example are analogous to the previous example. In this case, upon return from the ISGECAP service routine, the virtual storage area specified by OUTAREA@ contains the blocker report RIBs and RIBEs. Note that the program itself must reside in common area subpool 231, because the ISGECAP invocation is using the standard macro format.
Chapter 114. ISGENQ — Global resource serialization ENQ service

Description

Interface for Global Resource Serialization ENQ OBTAIN and RELEASE requests.

The GRS ENQ service routine is given control from the ISGENQ macro to:
- Obtain a single or multiple ENQs with or without associated device reserves.
- Change a single or multiple existing ENQs.
- Release a single or multiple ENQs.
- Test an obtain request.

This service is intended to replace ENQ, DEQ, and RESERVE.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state. Any PSW key</td>
</tr>
</tbody>
</table>

To use OWNINGTTOKEN, ENQMAX, or when the specified QNAME is one of the authorized QNAMEs, authorization must be one of the following: Supervisor state, PSW key 0-7, or APF authorized.

Note: When an authorized caller issues an OBTAIN request with an unauthorized QNAME, if COND=YES, the request is granted, but a warning return code and the reason ISGENQReturnUnprotectedQName are given. This is to warn that an unauthorized caller may block the ENQ, or even release the ENQ if running under the owning task. If COND=NO, authorized callers cannot obtain an ENQ on an unprotected resource.

- The authorized QNAMEs are:
  - ADRDFRAG
  - ADRDSN
  - ARCCENQG
  - BWODSN
  - SYSCTLG
  - SYSDSN
  - SYSIEA01
  - SYSIEECT
  - SYSIEFSD
  - SYSIGGV1
  - SYSIGGV2
  - SYSIPSWRD
  - SYSVSAM
  - SYSTOC
  - SYSZ*
ISGENQ macro

Environmental factor
Dispatchable unit mode:
Cross memory mode:
AMODE:
ASC mode:
Interrupt status:
Locks:
Control parameters:

Requirement
Task
Any PASN, any HASN, any SASN Note: The resulting ENQ is associated with the owning task in the home address space.
31- or 64-bit
Primary or access register (AR)
Enabled for I/O and external interrupts
The caller must not be locked.
Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).
The control parameters must be in the same key as the caller.
The ECB specified must be in the caller's home address space or in common.
The TCB of the owning task (the current task or specified by OWNINGTTOKEN) must be in the caller's home address space.
If a captured UCB address is specified, the captured UCB must be in the caller's home address space.

Programming requirements
The caller must include the ISGYCON macro to get the return and reason codes.
The caller must include the ISGYENQ macro to get the mappings for the ISGYENQAA, ISGYENQRES, ISGYENQTOKEN, and ISGYENQRETURN tables.
See "Avoiding Interlock" in z/OS MVS Programming: Assembler Services Guide to ensure that you are following the required protocols to prevent the interlock.

Restrictions
The caller must not have functional recovery routines (FRRs).

This macro supports multiple versions. Some keywords are unique to certain versions. See the "PLISTVER=IMPLIED_VERSION" on page 1142 parameter description.

Input register information
Before issuing the ISGENQ macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:
Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR15 is not 0</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.
ISGENQ macro

Syntax

main diagram

```
ISGENQ REQUEST=OBTAIN parameters-1
  ,COND=NO
ISGENQ REQUEST=CHANGE parameters-2
  ,COND=YES
ISGENQ REQUEST=RELEASE parameters-3
  ,COND=YES

```

parameters-1

```
ISGENQ name
  ,TEST=NO
  ,PARAMS=parameters-4
  ,COND=YES
  ,PARAMS=parameters-5
  ,COND=YES

```

parameters-2

```
ISGENQ MF=S
  ,TEST=NO
  ,PARAMS=parameters-6
  ,COND=YES
  ,PARAMS=parameters-6
  ,COND=YES

```

parameters-3

```
ISGENQ MF=(L,list addr,attr)
  ,TEST=NO
  ,PARAMS=parameters-6
  ,COND=YES
  ,PARAMS=parameters-6
  ,COND=YES

```

parameters-4

```
ISGENQ MF=(E,list addr,attr)
  ,TEST=NO
  ,PARAMS=parameters-6
  ,COND=YES
  ,PARAMS=parameters-6
  ,COND=YES

```

parameters-5

```
ISGENQ MF=(L,list addr,attr)
  ,TEST=NO
  ,PARAMS=parameters-6
  ,COND=YES
  ,PARAMS=parameters-6
  ,COND=YES

```

parameters-6

```
ISGENQ MF=(E,list addr,attr)
  ,TEST=NO
  ,PARAMS=parameters-6
  ,COND=YES
  ,PARAMS=parameters-6
  ,COND=YES

```

Parameters:

- **TEST**
  - NO
  - YES

- **ANSLEN**
  - NO_ANSLEN
  - ANSLEN

- **OWNINGTTOKEN**
  - CURRENT_TASK
  - OWNINGTTOKEN

- **RESLIST**
  - NO
  - YES

- **ENQTOKEN**
  - CURRENT_TASK
  - OWNINGTTOKEN

- **CONTROL**
  - EXCLUSIVE
  - SHARED

- **RETURNTABLE**
  - returntable

- **MF**
  - S
  - (L,list addr,attr)
  - (E,list addr,attr)
Chapter 114. ISGENQ — Global resource serialization ENQ service
ISGENQ macro

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the ISGENQ macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

 ANSAREA=ansarea
When TEST=YES and REQUEST=OBTAIN are specified, an optional output parameter, which contains the returned information. The area is a list of records mapped by ISGYNQAA in the ISGYENQ macro. For RESLIST=YES, the records are in the same order as the requests in the RESTABLE. ANSLEN is required if ANSAREA is specified.

Note: The answer area is returned only when RC=0 or RC=4.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

 ENQMAX=YES
 ENQMAX=NO
When TEST=NO and REQUEST=OBTAIN are specified, an optional parameter that indicates whether ENQMAX checking should be done. This keyword tells
global resource serialization whether a check is to be made to see if the limit for the number of concurrent resource requests has been exceeded. The default is ENQMAX=YES.

,ENQMAX=YES
Indicates ENQMAX checking should be done. IBM suggests that you use the default, ENQMAX=YES, to allow global resource serialization to perform this processing.

,ENQMAX=NO
Indicates that ENQMAX checking should not be used. Use ENQMAX=NO when you have a system-critical ENQ request that should be honored regardless of the concurrent number of resource requests made from the home address space.

Note: ENQMAX=NO can only be specified by an authorized requester and therefore can only override the maximum for authorized requesters.

See z/OS MVS Planning: Global Resource Serialization for more information.

,ANSLEN=anslen
,ANSLEN=NO_ANSLEN
When TEST=YES and REQUEST=OBTAIN are specified, an optional input parameter that is the length of the answer area provided. The answer area should be large enough to hold a ISGQENVQAA record and an RNAME for each request (specified by NUMRES, or one if RESLIST=NO). The maximum size area needed to contain one RNAME is 256 bytes. ANSAREA is required if ANSLEN is specified. The default is NO_ANSLEN.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,COND=NO
,COND=YES
An optional parameter that indicates how the request is handled for unsuccessful processing. The default is COND=NO.

,COND=NO
Indicates that if the request is not successful, then ISGENQ should ABEND the caller. COND=NO is mutually exclusive with RETCODE, RSNCODE, RETURNTABLE, WAITTYPE=ECB, and with TEST=YES.

,COND=YES
Indicates that ISGENQ should always return to the caller and indicate via return and reason codes whether the request was successful. If COND=YES is specified, RETCODE and RSNCODE (and RETURNTABLE, if RESLIST=YES) are required keywords.

Note: When COND=YES, ISGENQ tries to provide return and reason codes for the errors occurred during the process, though in some cases abends might be issued.

,CONTENTIONACT=WAIT
,CONTENTIONACT=FAIL
When TEST=NO and REQUEST=OBTAIN are specified, an optional parameter that indicates the action that should be taken if there is contention for the requested resource.

Note that a reserve request (where UCB@ is specified) that is not converted to only a global ENQ (Systems) will consist of an ENQ resource and a hardware reserve. For more information on reserve processing, see the description of the...
ISGENQ macro

"SYNCHRES=SYSTEM" on page 1148 keyword for more information on reserve processing. The default is CONTENTIONACT=WAIT.

,CONTENTIONACT=WAIT
  Indicates that the caller waits until the ENQ resource is available and, if applicable, the synchronous reserve I/O (see SYNCHRES) is complete.

,CONTENTIONACT=FAIL
  Indicates that if contention for the ENQ resource exists to cancel the ENQ obtain request and return to the caller.

Notes:
  • See CONTENTIONACT=WAIT with ECB@ as a means of timing the overall request.
  • For a reserve request (where UCB@ is specified), the ENQ resource is always obtained first. As such, CONTENTIONACT=FAIL indicates to cancel the entire request when there is contention on the ENQ resource. However, it does not apply to contention on the hardware reserve. See CONTENTIONACT=WAIT with WAITTYPE=ECB for information on how to manage or time hardware reserve contention.

,CONTROL=EXCLUSIVE
,CONTROL=SHARED
,CONTROL=VALUE
  When RESLIST=NO and REQUEST=OBTAIN are specified, a required parameter that is the control type of the ENQ to be obtained. If the resource is modified while under control of the task, the request must be for exclusive control. If the resource is not modified, the request should be for shared control.

,CONTROL=EXCLUSIVE
  Indicates that the request is for exclusive control of the resource.

,CONTROL=SHARED
  Indicates that the request is for shared control of the resource.

,CONTROL=VALUE
  the user provides a value, through the CONTROLVAL keyword, indicating the requested control.

,CONTROL=DO_NOT_OVERRIDE
,CONTROL=EXCLUSIVE
,CONTROL=SHARED
  When RESLIST=YES and REQUEST=OBTAIN are specified, an optional parameter that is the type of control to be used for all resources specified in the resource table. This overrides any control specified in the resource table. If the resource is modified while under control of the task, the request must be for exclusive control. If the resource is not modified, the request should be for shared control. The default is CONTROL=DO_NOT_OVERRIDE.

,CONTROL=DO_NOT_OVERRIDE
  Indicates that the control specified in the resource table should be used.

,CONTROL=EXCLUSIVE
  Indicates that all requests are for exclusive control of the resources.

,CONTROL=SHARED
  Indicates that all requests are for shared control of the resources.
When RESLIST=NO and REQUEST=CHANGE are specified, control is an optional keyword input that is the control type to which the ENQ is to be changed. If the resource is modified under control of the task the request must be for exclusive control. If the resource is not modified, the request should be for shared control. When RESLIST=YES is specified, all resources in the list will be changed to the specified scope. The default is CONTROL=EXCLUSIVE.

,CONTROL=EXCLUSIVE
Indicates that the request is to change to exclusive control of the resource.

,CONTROL=SHARED
Indicates that the request is to change to shared control of the resource.

,CONTROLVAL=controlval
When CONTROL=VALUE, RESLIST=NO and REQUEST=OBTAIN are specified, a required input parameter that contains a value indicating the desired control. The value provided must be equivalent to the constants provided in the ISGYENQ macro indicating the control. (See the ISGYENQ_aControl constants in the ISGYENQ macro for more information.)

To code: Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

,ECB=ecb@  
When WAITTYPE=ECB, CONTENTIONACT=WAIT, TEST=NO and REQUEST=OBTAIN are specified, a required input parameter that contains the address of the ECB to be posted when the requested resource(s) is/are obtained.

The ECB must be in one of the following locations:
• the home address space of the caller.
• common space.
• for unauthorized requesters, in the same storage key as the requester.

When the ISGENQ service returns to the caller, the return and reason codes specify for each resource whether the task has been given control of the resource or needs to wait for the ECB to be posted.

When the ECB is posted, it contains a return/reason code pair. Bits 8-23 contain the low-order halfword of the reason code and bits 24-31 contain the low-order byte of the return code. For a RESLIST=NO request, the ECB contains the return and reason code for the request. For a RESLIST=YES request, the ECB contains an overall return code.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,ENQTOKEN=enqtoken
When RESLIST=NO and REQUEST=OBTAIN are specified, a required output parameter that is a token that uniquely identifies the ENQ. The ENQTOKEN is used on subsequent REQUEST=RELEASE or CHANGE invocations to release or change the ENQ request.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,ENQTOKEN=enqtoken
When RESLIST=NO and REQUEST=CHANGE are specified, a required input parameter that is an ENQ Token of the ENQ to be changed.
ISGENQ macro

**To code:** Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

`,ENQTOKEN=enqtoken`

When RESLIST=NO and REQUEST=RELEASE are specified, a required input parameter that is an ENQ Token of the ENQ to be released.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

`,ENQTOKENTBL=enqtokentbl`

When RESLIST=YES and REQUEST=OBTAIN are specified, a required output parameter that is a table of ENQ tokens. Mapped by ISGYENQToken in the ISGENQ macro. To easily release any ENQs obtained by a REQUEST=OBTAIN use the same ENQToken table as input to a REQUEST=RELEASE.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a character field.

`,ENQTOKENTBL=enqtokentbl`

When RESLIST=YES and REQUEST=CHANGE are specified, a required input parameter that is a table of ENQ Tokens. Mapped by ISGYENQToken in the ISGENQ macro.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a character field.

`,ENQTOKENTBL=enqtokentbl`

When RESLIST=YES and REQUEST=RELEASE are specified, a required input parameter that is a table of ENQ Tokens. Mapped by ISGYENQToken in the ISGENQ macro.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a character field.

`,MF=S`

`,`MF=(L,list addr)`

`,`MF=(L,list addr,attr)`

`,`MF=(L,list addr,0D)`

`,`MF=(E,list addr)`

`,`MF=(E,list addr,COMPLETE)`

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NUMRES=numres
When RESLIST=YES and REQUEST=OBTAIN are specified, a required input parameter that is the number of resource entries in the resource table. The specified value can be in the range of 1 to 2^6-1 (65535).

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

,NUMRES=numres
When RESLIST=YES and REQUEST=CHANGE are specified, a required input parameter that is the number of ENQ tokens in the ENQ token table. The specified value can be in the range of 1 to 2^6-1 (65535).

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

,NUMRES=numres
When RESLIST=YES and REQUEST=RELEASE are specified, a required input parameter that is the number of ENQ tokens in the ENQ Token Table. The specified value can be in the range of 1 to 2^6-1 (65535).

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

,OWNINGTTOKEN=owningttoken

,OWNINGTTOKEN=CURRENT_TASK
When WAITTYPE=ECB, CONTENTIONACT=WAIT, TEST=NO and REQUEST=OBTAIN are specified, an optional input parameter that is the task token (TToken) of the task on whose behalf the ENQ is to be obtained. The TToken must specify a task in the caller’s home address space.

Note: Mutually exclusive with RESERVEVOLUME=YES. The default is CURRENT_TASK.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,OWNINGTTOKEN=owningttoken

,OWNINGTTOKEN=CURRENT_TASK
When CONTENTIONACT=FAIL, TEST=NO and REQUEST=OBTAIN are specified, an optional input parameter that is the task token (TToken) of the task on whose behalf the ENQ is to be obtained. The TToken must specify a task in the caller’s home address space.

Note: Mutually exclusive with RESERVEVOLUME=YES. The default is CURRENT_TASK.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.
When TEST=YES and REQUEST=OBTAIN are specified, an optional input parameter that is the task token (TToken) of the task on whose behalf the test request is to be performed. The TToken must specify a task in the caller's home address space.

Note: Mutually exclusive with RESERVEVOLUME=YES. The default is CURRENT_TASK.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

When REQUEST=CHANGE is specified, an optional input parameter that is the task token (TToken) of the task that owns the ENQ that is to be changed. The TToken must specify a task in the caller's home address space. The default is CURRENT_TASK.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

When REQUEST=RELEASE is specified, an optional input parameter that is the task token (TToken) of the task that owns the ENQs that are to be released. The TToken must specify a task in the caller's home address space. The default is CURRENT_TASK.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- IMPLIED_VERSION, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- MAX, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- 1, which supports all parameters except those specifically referenced in higher versions.
- 2, which supports both the following parameters and those from version 1: USERDATA
To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1, or 2

\texttt{,QNAME=qname}

When RESLIST=NO and REQUEST=OBTAIN are specified, a required input parameter that is the QNAME of the resource. The QNAME can contain any character from X'00' to X'FF'. However, a unique readable value that identifies the functional area or a high level of what is being serialized is preferred. Every program issuing a request for a serially reusable resource must use the same QNAME, RNAME, and Scope to represent the resource. Some names, such as those beginning with certain letter combinations (SYSZ for example), are used to protect system resources by requiring that the issuing program be in supervisor state, or system key, or APF-authorized. Authorized programs must use a restricted QNAME (as described under Minimum authorization in the Environment section for this service) to prevent interference from unauthorized programs.

For a list of QNAME (also known as major name) and RNAME (also known as minor name) ENQ or DEQ names and the resources that issue the ENQ or DEQ, see z/OS MVS Diagnosis: Reference.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

\texttt{,QNAME=DO\_NOT\_OVERRIDE}

When RESLIST=YES and REQUEST=OBTAIN are specified, an optional input parameter that is a common QNAME to be used for all resources in the resource table. This overrides any QNAMEs specified in the resource table. The QNAME can contain any character from X'00' to X'FF'. However, a unique readable value that identifies the functional area or a high level of what is being serialized is preferred. Every program issuing a request for a serially reusable resource must use the same QNAME, RNAME, and Scope to represent the resource. Some names, such as those beginning with certain letter combinations (SYSZ for example), are used to protect system resources by requiring that the issuing program be in supervisor state, or system key, or APF-authorized. Authorized programs must use a restricted QNAME (as described under Minimum authorization in the Environment section for this service) to prevent interference from unauthorized programs.

For a list of QNAME (also known as major name) and RNAME (also known as minor name) ENQ or DEQ names and the resources that issue the ENQ or DEQ, see z/OS MVS Diagnosis: Reference.

The default is DO\_NOT\_OVERRIDE.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

\texttt{REQUEST=OBTAIN}
\texttt{REQUEST=CHANGE}
\texttt{REQUEST=RELEASE}

A required parameter that indicates the type of ISGENQ request.

\texttt{REQUEST=OBTAIN}

Indicates a request to obtain an ENQ for a resource.
ISGENQ macro

REQUEST=CHANGE
   Indicates a request to change the status an ENQ from shared to exclusive control.

REQUEST=RELEASE
   Indicates a request to release (dequeue) the ENQ for a resource.

RESERVEVOLUME=NO
RESERVEVOLUME=YES
   When RESLIST=NO and REQUEST=OBTAIN are specified, an optional parameter. The default is RESERVEVOLUME=NO.

RESERVEVOLUME=NO
   Indicates to issue a normal ENQ obtain and not a reserve.

RESERVEVOLUME=YES
   Indicates that after the ENQ resource is obtained that a reserve for the given device (shared DASD) is to be issued.
   Note: RESERVEVOLUME=YES is mutually exclusive with OWNINGTTOKEN.

RESLIST=NO
RESLIST=YES
   When REQUEST=OBTAIN is specified, an optional parameter. The default is RESLIST=NO.

RESLIST=NO
   Indicates to obtain an ENQ for a single resource.

RESLIST=YES
   Indicates to obtain ENQs for multiple resources specified in a resource table. Specifying multiple requests in a list ensures that they are processed atomically with respect to other ISGENQ requests. However, the order in which the requests are processed is unpredictable. Each request is treated as a separate request, and if COND=YES is specified, then the return code for each request should be checked.
   Note: An easy way to release a list of ENQs is to use the output ENQTOKEN table from the OBTAIN request as input to a RELEASE request.

RESLIST=NO
RESLIST=YES
   When REQUEST=CHANGE is specified, an optional parameter. The default is RESLIST=NO.

RESLIST=NO
   Indicates to change the control of a single ENQ.

RESLIST=YES
   Indicates to change the control for multiple ENQs.

RESLIST=NO
RESLIST=YES
   When REQUEST=RELEASE is specified, an optional parameter. The default is RESLIST=NO.

RESLIST=NO
   Indicates to single ENQ RELEASE request.

RESLIST=YES
   Indicates to change the disposition for multiple ENQs.
ISGENQ macro

Note: A easy way to release a list of ENQs is to use the output
ENQTOKEN table from the OBTAIN request as input to a RELEASE
request.

,RESTABLE=restable
When RESLIST=YES and REQUEST=OBTAIN are specified, a required input
parameter that is a table specifying multiple ENQ requests. The resource table
is mapped by ISGYENQRes in the ISGYENQ macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a
character field.

,RETCODE=retcode
An optional output parameter into which the return code is to be copied from
GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without
parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or
(15), (GPR15), (REG15), or (R15).

,RETURNTABLE=returntable
When RESLIST=YES and REQUEST=OBTAIN are specified, an optional output
parameter that is a table that contains the return and reason codes. Mapped by
ISGYENQReturn in the ISGYENQ macro. The return table is only valid when
ISGENQRsn_NonZeroReturnCodes is returned in the RSNCODE. Mutually
exclusive with COND=NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a
character field.

,RETURNTABLE=returntable
When RESLIST=YES and REQUEST=CHANGE are specified, an optional output
parameter that is a table that contains the return and reason codes. Mapped by
ISGYENQReturn in the ISGYENQ macro. The return table is only valid when
ISGENQRsn_NonZeroReturnCodes is returned in the RSNCODE. Mutually
exclusive with COND=NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a
character field.

,RETURNTABLE=returntable
When RESLIST=YES and REQUEST=RELEASE are specified, an optional output
parameter that is a table that contains the return and reason codes. Mapped by
ISGYENQReturn in the ISGYENQ macro. The return table is only valid when
ISGENQRsn_NonZeroReturnCodes is returned in the RSNCODE. Mutually
exclusive with COND=NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a
character field.

,RNAME=rname
When RESLIST=NO and REQUEST=OBTAIN are specified, a required input
parameter that is the RNAME for the resource. The RNAME must be from 1 to
255 bytes long, and can contain any hexadecimal character from X'00' to X'FF'.

To code: Specify the RS-type address, or address in register (2)-(12), of a
character field.

,RNAME=rname
When RESLIST=NO and REQUEST=OBTAIN are specified, a required input
parameter that is the common RNAME to be used for all resources in the

Chapter 114. ISGENQ — Global resource serialization ENQ service  1145
resource table. This overrides any RNAMEs specified in the resource table. The RNAME must be from 1 to 255 bytes long, and can contain any hexadecimal character from X'00' to X'FF'. The default is DO_NOT_OVERRIDE.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a character field.

```
,,RNAMELEN=rnamelen
```

When RESLIST=NO and REQUEST=OBTAIN are specified, a required input parameter that is the length of the given RNAME. The specified length can be in the range of 1 to 255.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

```
,,RNAMELEN=rnamelen
,,RNAMELEN=DO_NOT_OVERRIDE
```

When RESLIST=YES and REQUEST=OBTAIN are specified, an optional input parameter that is a common length to be used for all RNAMEs in the resource table, or if a common RNAME is specified, it is the length of the common RNAME. The specified length can be in the range of 1 to 255. This overrides any RNAMEs lengths specified in the resource table. The default is DO_NOT_OVERRIDE.

**To code:** Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

```
,,RNL=YES
,,RNL=NO
```

When RESERVEVOLUME=NO, RESLIST=NO and REQUEST=OBTAIN are specified, an optional parameter that indicates whether the scope can be changed by global resource serialization resource name list (RNL) processing or the dynamic exits. The default is RNL=YES.

```
,,RNL=DO_NOT_OVERRIDE
```

Indicates that the RNL specifications in the resource table should be used.
Indicates that global resource serialization RNL processing should be used, which can cause the scope of a resource to change. IBM suggests that you use the default, RNL=YES, to allow global resource serialization to perform RNL processing.

Indicates that global resource serialization RNL processing should not be used. The scope of the resource cannot be changed by the RNLs or any dynamic exits. Use RNL=NO when you are sure that you want the request to be processed only by global resource serialization using only the specified scope. When RNL=NO is specified, the ENQ request is ignored by alternative serialization products.

An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), (REG0), (REG00), or (R0).

When RESERVEVOLUME=NO, RESLIST=NO and REQUEST=OBTAIN are specified, a required parameter that is the scope of the resource.

Indicates that the resource is serialized only within an address space. If STEP is specified, a request for the same QNAME and RNAME from a program in another address space denotes a different resource.

Indicates that the resource is serialized across all address spaces in a system.

Indicates that the resource is serialized across all systems in a GRS Star or GRS Ring complex.

Indicates that the resource is serialized across all systems in a GRS Star sysplex or GRS ring. (Same as scope SYSTEMS.)

the user provides a value, through the SCOPEVAL keyword, indicating the requested scope.

When RESLIST=YES and REQUEST=OBTAIN are specified, an optional parameter that is the scope to be used for all resources in the resource table. This overrides any scopes specified in the resource table. The default is SCOPE=DO_NOT_OVERRIDE.
ISGENQ macro

,SCOPE=DO_NOT_OVERRIDE
Indicates that the scope specified in the resource table should be used.

,SCOPE=STEP
Indicates that the resource is serialized only within an address space. If
STEP is specified, a request for the same QNAME and RNAME from a
program in another address space denotes a different resource.

,SCOPE=SYSTEM
Indicates that the resource is serialized across all address spaces in a
system.

,SCOPE=SYSTEMS
Indicates that the resource is serialized across all systems in a GRS Star or
GRS Ring complex.

,SCOPE=SYSPLEX
Indicates that the resource is serialized across all systems in a GRS Star
sysplex or GRS ring. (Same as scope SYSTEMS.)

,SCOPEVAL=scopeval
When SCOPE=VALUE, RESERVEVOLUME=NO, RESLIST=NO and
REQUEST=OBTAIN are specified, a required input parameter that contains a
value indicating the desired scope. The value provided must be equivalent to
the constants provided in the ISGYENQ macro indicating the scope. (See the
ISGYENQ_ constants in the ISGYENQ macro for more information.)

To code: Specify the RS-type address, or address in register (2)-(12), of an
one-byte field.

,SYNCHRES=SYSTEM
,SYNCHRES=YES
,SYNCHRES=NO
When RESERVEVOLUME=YES, RESLIST=NO and REQUEST=OBTAIN are
specified, an optional parameter that specifies whether the request should issue
a synchronous reserve. A synchronous reserve immediately reserves the
volume instead of waiting for the first use.

Note that an RC=4 (ISGENQRc_Warn), RSC=0403
(ISGENQRsn_ECBWillBePosted) is presented for CONTENTIONACT=WAIT,
WAITTYPE=ECB, reserve requests (where UCB@ is specified) when there is
contention on the ENQ resource or there was no contention on the resource,
and the reserve I/O was done synchronously. The default is
SYNCHRES=SYSTEM.

,SYNCHRES=SYSTEM
Indicates that the installation system default SYNCHRES setting should be
used.

,SYNCHRES=YES
Indicates to issue a synchronous reserve. In cases where the hardware
reserve is performed (it was not converted to a Global/Systems ENQ), the
caller is ensured that the reserve I/O is complete when the ISGENQ
request has successfully completed.

,SYNCHRES=NO
Indicates that a synchronous reserve should be avoided when possible.
Some devices require that the reserve must be done synchronously
regardless of this setting. If the reserve I/O is not done synchronously, the
reservce is done when the first I/O is done to the device after the reserve request is issued. For more information, see z/OS MVS Planning: Global Resource Serialization.

SYNCHRES=DO_NOT_OVERRIDE
SYNCHRES=SYSTEM
SYNCHRES=YES
SYNCHRES=NO

When RESLIST=YES and REQUEST=OBTAIN are specified, an optional parameter that specifies whether all requests specified in the resource table should issue a synchronous reserve. This overrides any SYNCHRES specified in the resource table. A synchronous reserve immediately reserves the volume instead of waiting for the first use. The default is SYNCHRES=DO_NOT_OVERRIDE.

SYNCHRES=DO_NOT_OVERRIDE
Indicates that the SYNCHRES specified in the resource table should be used.

SYNCHRES=SYSTEM
Indicates that the system default setting should be used.

SYNCHRES=YES
Indicates to issue a synchronous reserve. In cases where the the hardware reserve is performed (it was not converted to a Global/Systems ENQ), the caller is ensured that the reserve I/O is complete when the request has successfully completed.

SYNCHRES=NO
Indicates that a synchronous reserve should be avoided when possible. Some devices require that the reserve must be done synchronously regardless of this setting. If the reserve I/O is not done synchronously, the reserve is done when the first I/O is done to the device after the reserve request is issued. See z/OS MVS Planning: Global Resource Serialization for more information.

TEST=NO
TEST=YES

When REQUEST=OBTAIN is specified, an optional parameter. The default is TEST=NO.

TEST=NO
Indicates that this is not a test request. The ENQ must be obtained.

TEST=YES
Indicates that this is a test request. The ENQ must not be obtained. This parameter setting can be used to obtain information about how the given obtain request is processed and how a resource is currently held by the current task or a task specified by OWNINGTTOKEN.

Mutually exclusive with COND=NO.

For existing requests from the same task, which match the specified resource, the ENQToken of that request is returned.

See ISGQUERY SEARCH=BY_ENQTOKEN for information about outstanding ENQ requests.

The following return and reason codes can be used to determine if the resource is available and how it might be held by the OWNINGTTOKEN task:
- ISGENQRc_ok
**ISGENQ macro**

- ISGENQRsn_NotImmediatelyAvailable
- ISGENQRsn_TaskOwnsExclusive
- ISGENQRsn_TaskOwnsShared
- ISGENQRsn_TaskWaiting

**UCB=ucb@**

When RESERVEVOLUME=YES, RESLIST=NO and REQUEST=OBTAIN are specified, a required input parameter that contains the address of the UCB for the device to be reserved. For unauthorized callers, the UCB must be allocated to the job step before ISGENQ RESERVEVOLUME(YES) is issued.

Note: Authorized callers do not need to allocate the UCB to the job step before invoking ISGENQ, but the caller must serialize the UCB against dynamic I/O reconfiguration requests. The caller can accomplish this serialization by allocating or pinning the UCB. Such serialization ensures that a dynamic I/O reconfiguration request does not delete or reuse the UCB before the ISGENQ macro uses the address.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

**UCB=DO_NOT_OVERRIDE**

When RESLIST=YES and REQUEST=OBTAIN are specified, an optional input parameter that contains the address of the UCB for the device to be reserved for all resources in the resource table. This overrides any UCB addresses specified in the resource table. The default is DO_NOT_OVERRIDE.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

**USERDATA=userdata**

**USERDATA=NO_USERDATA**

When TEST=NO and REQUEST=OBTAIN are specified, an optional input parameter that contains the user data to be associated with this request. For information about using USERDATA as a filter, or making ISGQUERY return USERDATA for requests, see Chapter 127, “ISGQUERY — Global resource serialization query service,” on page 1237.

Note that GRS has no interests in the contents of the USERDATA. Unlike the QNAME, RNAME, and SCOPE parameters, USERDATA has no meaning in the definition of the logically serialized resource identity. For example, exclusive requests with different user data and the same QNAME, RNAME, and SCOPE contend with each other.

This request requires a version 2 parameter list. The default is NO_USERDATA.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

**WAITTYPE=SUSPEND**

**WAITTYPE=ECB**

When CONTENTIONACT=WAIT, TEST=NO and REQUEST=OBTAIN are specified, an optional parameter that indicates the method by which the caller waits. The default is WAITTYPE=SUSPEND.

**WAITTYPE=SUSPEND**

Indicates that the current task is suspended until the entire request is completed.
```
\textbf{WAITTYPE=ECB}

Indicates that if contention for the ENQ resource exists or the device reserve is done synchronously (see \textit{"\texttt{SYNCHRES=SYSTEM}" on page 1148}), return to the caller, and post the ECB when the request is complete.

Mutually exclusive with COND=NO.

\textit{WAITTYPE=ECB} in combination with setting a timer with ECB can be used to control the amount of time that you are willing to wait for either ENQ contention or a synchronous reserve to complete. If the request does not complete before the time expires you can do the following actions.

\begin{itemize}
  \item You can use the the ISGCA and ISGQUERY services to interrogate the overall state of the request and associated resource.
  \item You can back out of the request using an ISGENQ REQUEST\textbullet;RELEASE request.
\end{itemize}

\textbf{ABEND codes}

For REQUEST\textbullet;OBTAIN and REQUEST\textbullet;CHANGE requests the caller might encounter abend codes X'138', X'238', X'338', X'438', X'538', X'638', X'738', X'838', X'938'.

For REQUEST\textbullet;RELEASE requests the caller might encounter abend codes X'130', X'230', X'330', X'430', X'530', X'630', X'730', X'830', X'930'.

For explanations and responses for these codes, see \url{z/OS MVS System Codes}.

Note that the ABEND reason codes correspond to the same reason codes listed in Table 100.

\textbf{Return and reason codes}

When the ISGENQ macro returns control to your program:

\begin{itemize}
  \item GPR 15 (and \texttt{retcode}, when you code RETCODE) contains a return code.
  \item When the value in GPR 15 is not zero, GPR 0 (and \texttt{rsnrcode}, when you code RSNCODE) contains a reason code.
\end{itemize}

Macro ISGYCON provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support the xxxx value, where xxxx represent 4 hex digits. Note that when the xxxx value is 'E0F2' hexadecimal, it indicates a reason-code set by the ISGNQXITBATCH or ISGNQXITBATCHCND exits.

\begin{table}[h]
\centering
\caption{Return and Reason Codes for the ISGENQ Macro}
\begin{tabular}{|c|c|c|}
\hline
\textbf{Return Code} & \textbf{Reason Code} & \textbf{Equate Symbol Meaning and Action} \\
\hline
00 & — & \textbf{Equate Symbol: ISGENQrc_OK} \\
   &  & \textbf{Meaning:} ISGENQ request successful. Depending on the type of request, the ENQ is successfully obtained, changed to exclusive, or released. If RESLIST=YES is specified, all ENQ obtain, change, and release requests are successful. For REQUEST\textbullet;OBTAIN, TEST\textbullet;YES, the resource is immediately available.
   &  & \textbf{Action:} None required.
\hline
\end{tabular}
\end{table}

\url{z/OS MVS System Codes}
```
### Table 100. Return and Reason Codes for the ISGENQ Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>—</td>
<td><strong>Equate Symbol</strong>: ISGENQRc_Warn</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: Warning</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Refer to action under the individual reason code.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0401</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_NonZeroReturnCodes</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: A non-zero return code was issued for one or more entries in a RESLIST=YES request. The return table has the return and reason codes for each of the requests in the list.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: See the return and reason codes returned in the RETURNTABLE.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0402</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_RequestNotProcessed</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: For RESLIST=YES requests. One of the other requests in the RESTABLE failed such that this request was prevented from being processed. Note that requests in a RESTABLE are not necessarily processed in the order they appear in the RESTABLE. Note: This reason code returned only in the RETURNTABLE, not through the RSNCODE keyword.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Check the return and reason codes for all other requests in the RETURNTABLE to identify the problem.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0403</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_ECBWillBePosted</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: For REQUEST=OBTAIN CONTENTIONACT=WAIT WAITTYPE=ECB, the OBTAIN request was successful, but the ENQ resource was not immediately available or the reserve I/O needed to be done synchronously (SYNCHRES). The ECB is posted when all requested resources are owned by the specified task, or when an error has occurred. The ENQToken for the request has been returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Wait on the ECB and check the return code in the ECB before using the requested resources.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0404</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_NotImmediatelyAvailable</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: The ENQ of the resource was not immediately available. For REQUEST=OBTAIN CONTENTIONACT=FAIL, the requested resource is not obtained. For REQUEST=OBTAIN TEST=YES, the holder is a task other than OWNINGTTOKEN.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: No action required.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0405</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_TaskOwnsExclusive</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: For REQUEST=OBTAIN, including TEST=YES, the given task specified by OWNINGTTOKEN already owns the specified resource exclusively. The ENQToken for the owning request has been returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: No action required.</td>
</tr>
</tbody>
</table>
### Table 100. Return and Reason Codes for the ISGENQ Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04          | xxxx0406    | **Equate Symbol**: ISGENQ_Rsn_TaskOwnsShared | **Meaning**: For a REQUEST=OBTAIN, including TEST=YES, the given task specified by OWNINGTTOKEN already owns the specified resource shared. The ENQToken for the owning request has been returned.  
**Action**: No action required. |
| 04          | xxxx0407    | **Equate Symbol**: ISGENQ_Rsn_TaskWaiting | **Meaning**: For a REQUEST=OBTAIN, including TEST=YES, the given task specified by OWNINGTTOKEN is already waiting for control of the specified resource. The ENQToken for the waiting request has been returned.  
**Action**: No action required. |
| 04          | xxxx0409    | **Equate Symbol**: ISGENQ_Rsn_OtherSharedOwners | **Meaning**: For REQUEST=CHANGE. The control cannot be changed to exclusive. There are other shared owners of the resource.  
**Action**: No action required. |
| 04          | xxxx040A    | **Equate Symbol**: ISGENQ_Rsn_TaskDoesNotOwn | **Meaning**: For REQUEST=CHANGE. The control cannot be changed to exclusive. The task does not yet own the resource.  
**Action**: No action required. |
| 04          | xxxx040B    | **Equate Symbol**: ISGENQ_Rsn_TaskSuspendedForResource | **Meaning**: For REQUEST=RELEASE. The task that requested the ENQ obtain has not yet been assigned control of the resource. The task continues waiting and the resource is not released. (This reason code might result in an exit routine, which received control because of an interruption, issued a RELEASE request on behalf of the task.)  
**Action**: Correct the program so that the ISGENQ RELEASE request is issued only after the ISGENQ OBTAIN request has returned to the task. If possible, avoid issuing the RELEASE request in the exit routine. |
### ISGENQ macro

#### Table 100. Return and Reason Codes for the ISGENQ Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04          | xxxx040D    | Equate Symbol: ISGENQRsn_UnprotectedQName | **Meaning:** For REQUEST=OBTAIN. An authorized caller requested an ENQ with an unauthorized QNAME.  
For TEST=NO,COND=YES, the OBTAIN request completed successfully, an unauthorized caller under the same owning task might release the ENQ. The ENQToken has been returned.  
For TEST=NO, COND=NO, the requester was abended with a X'438' abend. The request might not have completed successfully  
For TEST=YES requests, the resource is currently available.  
**Action:** No action required. If the ENQ needs to be protected from unauthorized RELEASE requests or from unauthorized callers obtaining an ENQ to block this request, specify one of the authorized QNAMEs for the resource. |
| 04          | xxxx040E    | Equate Symbol: ISGENQRsn_UnprotectedExitQNAME | **Meaning:** For REQUEST=OBTAIN. An authorized caller requested an ENQ with a QNAME that a dynamic exit changed to an unauthorized QNAME. For TEST=NO, the OBTAIN request completed successfully, an unauthorized caller under the same owning task might release the ENQ. The ENQToken has been returned. For TEST=YES requests, the resource is currently available but the QNAME was changed by a dynamic exit to an unprotected QNAME.  
**Action:** No action required. Contact the system programmer, if the ENQ needs to be protected from unauthorized RELEASE requests or from unauthorized callers obtaining an ENQ to block this request. The system programmer should check the ISGNQXIT installation exits to ensure that they are not coded to specify an unauthorized QNAME for authorized requests. |
| 04          | xxxx040F    | Equate Symbol: ISGENQRsn_ECBAtleastOneRequestFailed | **Meaning:** For REQUEST=OBTAIN RESLIST=Yes with ECB@, at least one request failed to be processed. Some requests might have been processed unsuccessfully. The system might not backout any successfully processed requests.  
**Note:** This reason code is returned in a posted ECB, not through the RSNCODE or RETURNTABLE keywords.  
**Action:** The user should issue an ISGQUERY on the ENQTOKENs to see if they were obtained and take appropriate action.  
Alternately, the user can release all the ENQs with a ISGENQ REQUEST=RELEASE with ENQTOKENTBL and reissue the ISGENQ OBTAIN request. |
| 08          | —           | Equate Symbol: ISGENQRc_ParmError | **Meaning:** ISGENQ request specified parameters in error.  
**Action:** Refer to action under the individual reason code. |
**Table 100. Return and Reason Codes for the ISGENQ Macro (continued)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>xxxx0801</td>
<td>Equate Symbol: ISGENQRsn_BadPlistAddress</td>
<td><strong>Meaning:</strong> Unable to access parameter list.</td>
<td><strong>Action:</strong> Check that the entire parameter list is addressable. If in AR-mode, check that the ALET of the parameter list is correct. Note that if this macro is issued in AR-mode, SYSSTATE ASCENV=AR must be issued before this macro. Ensure that the storage is in the same key as the caller.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0802</td>
<td>Equate Symbol: ISGENQRsn_BadPlistALET</td>
<td><strong>Meaning:</strong> Bad parameter list ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller’s Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.</td>
<td><strong>Action:</strong> Ensure that the ALET of the parameter list is valid. Its access register may not have been set up properly.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0803</td>
<td>Equate Symbol: ISGENQRsn_BadPlistVersion</td>
<td><strong>Meaning:</strong> Bad parameter list version number. The service level of GRS on which the caller is running does not support this version of the ISGENQ service, or the ISGENQ parameter list version is lower than the minimum required for parameters that were specified.</td>
<td><strong>Action:</strong> Check for possible storage overlay of the parameter list. Retry the request with the correct version number. Verify that your program was assembled with the correct macro library for the release of MVS on which your program is running.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0804</td>
<td>Equate Symbol: ISGENQRsn_ReservedFieldNotNull</td>
<td><strong>Meaning:</strong> A reserved field in the parameter list is non-zero.</td>
<td><strong>Action:</strong> Check for possible storage overlay of the parameter list.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0805</td>
<td>Equate Symbol: ISGENQRsn_MutuallyExclusive</td>
<td><strong>Meaning:</strong> Mutually exclusive keywords were specified.</td>
<td><strong>Action:</strong> Check for a possible storage overlay of the parameter list.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0806</td>
<td>Equate Symbol: ISGENQRsn_BadRequest</td>
<td><strong>Meaning:</strong> Bad REQUEST parameter.</td>
<td><strong>Action:</strong> IBM suggests that the ISGENQ macro is used when invoking the ISGENQ service.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0807</td>
<td>Equate Symbol: ISGENQRsn_BadContentionAct</td>
<td><strong>Meaning:</strong> Bad CONTENTIONACT parameter.</td>
<td><strong>Action:</strong> Check for possible storage overlay of the parameter list.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0808</td>
<td>Equate Symbol: ISGENQRsn_BadOwningTToken</td>
<td><strong>Meaning:</strong> The specified TToken does not represent a valid task.</td>
<td><strong>Action:</strong> Ensure that the task token (TToken) represents a valid task.</td>
</tr>
</tbody>
</table>
### Table 100. Return and Reason Codes for the ISGENQ Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0809    | Equate Symbol: ISGENQRSn_BadAnsAreaAddress | Meaning: Unable to access the answer area.  
Action: Ensure that the entire answer area is addressable. Ensure that the storage is in the same key as the caller. |
| 08          | xxxx080A    | Equate Symbol: ISGENQRSn_BadAnsAreaALET | Meaning: Bad answer area ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller’s Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.  
Action: Ensure that the ALET of the answer area is valid. Its access register may not have been set up properly. |
| 08          | xxxx080B    | Equate Symbol: ISGENQRSn_AnsLenTooSmall | Meaning: The specified answer area length was too small to return the requested information.  
Action: Invoke ISGENQ again with a larger answer area. The answer area length needed is dependent on the number of resource requests specified in NUMRES. |
| 08          | xxxx080C    | Equate Symbol: ISGENQRSn_BadRNameAddress | Meaning: Unable to access the RNAME.  
Action: Ensure that the entire RNAME is addressable. Ensure that the storage is in the same key as the caller. |
| 08          | xxxx080D    | Equate Symbol: ISGENQRSn_BadRNameALET | Meaning: Bad RNAME ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller’s Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.  
Action: Ensure that the ALET of the RNAME is valid. Its access register may not have been set up properly. |
| 08          | xxxx080E    | Equate Symbol: ISGENQRSn_BadRNameLen | Meaning: The RNAME length specified is not valid.  
Action: Ensure the RNAME length field contains a number in the range of 1-255. |
| 08          | xxxx080F    | Equate Symbol: ISGENQRSn_BadScope | Meaning: Bad SCOPE keyword parameter.  
Action: Check for possible storage overlay of the parameter list. |
### Table 100. Return and Reason Codes for the ISGENQ Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>xxxx0810</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadUCB@</td>
<td>The storage specified by the UCB@ keyword does not map to a valid UCB.</td>
<td>Ensure that the UCB@ points to a valid UCB.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0811</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadCond</td>
<td>Bad COND keyword parameter.</td>
<td>IBM suggests that the ISGENQ macro is used when invoking the ISGENQ service.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0812</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadSynchRes</td>
<td>Bad SYNCHRES keyword parameter.</td>
<td>Check for possible storage overlay of the parameter list.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0813</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadENQTokenAddress</td>
<td>Unable to access the ENQToken.</td>
<td>Ensure that the entire ENQToken is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Ensure that the storage is in the same key as the caller. Note: The ISGENQ request might not have completed.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0814</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadENQTokenALET</td>
<td>Bad ENQToken ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller’s Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.</td>
<td>Ensure that the ALET of the ENQToken is valid. Its access register may not have been set up properly. Note: The ISGENQ request might not have completed.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0815</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadENQToken</td>
<td>For REQUEST=RELEASE or REQUEST=CHANGE, the specified ENQToken does not represent an ENQ for the given task (current task or specified by OWNINGTTOKEN).</td>
<td>Ensure that the specified ENQToken is from a previous request for the given task, that has not been subsequently released.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0816</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadNumRes</td>
<td>The NUMRES specified is not valid.</td>
<td>Ensure the NUMRES field contains a number in the range of 1-65535 (2^16-1)</td>
</tr>
</tbody>
</table>
ISGENQ macro

Table 100. Return and Reason Codes for the ISGENQ Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>xxxx0817</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadResTableAddress</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Unable to access the resource table.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Ensure that the entire resource table is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Check that the resource table length is correct. Ensure that the storage is in the same key as the caller.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx0818</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadResTableALET</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Bad resource table ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller’s Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Ensure that the ALET of the resource table is valid. Its access register may not have been set up properly.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx0819</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadResTable</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: The RESTABLE specified is not valid.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Ensure that the resource table does not specify mutually exclusive parameters.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx081A</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadENQTokenTblAddress</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Unable to access the ENQToken table.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Ensure that the entire ENQToken table is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Check that the ENQToken table length is correct. Ensure that the storage is in the same key as the caller. Note: The ISGENQ request might not have completed.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx081B</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadENQTokenTblALET</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Bad ENQToken table ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller’s Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Ensure that the ALET of the ENQToken table is valid. Its access register may not have been set up properly. Note: The ISGENQ request might not have completed.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx081C</td>
<td><strong>Equate Symbol</strong>: ISGENQRsn_BadReturnTableAddress</td>
</tr>
<tr>
<td></td>
<td><strong>Meaning</strong>: Unable to access the return table.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Ensure that the entire return table is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Check that the return table length is correct. Ensure that the storage is in the same key as the caller. Note: The ISGENQ request might not have completed.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 100. Return and Reason Codes for the ISGENQ Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx081D    | **Equate Symbol:** ISGENQRsn_BadReturnTableALET  
**Meaning:** Bad return table ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.  
**Action:** Ensure that the ALET of the return table is valid. Its access register may not have been set up properly. Note: The ISGENQ request might not have completed. |
| 08          | xxxx081E    | **Equate Symbol:** ISGENQRsn_NotAuthorizedForQName  
**Meaning:** For REQUEST=OBTAIN. An unauthorized caller specified an authorized QNAME.  
**Action:** Unauthorized callers must avoid specifying the authorized QNAMEs listed in the ISGENQ macro prologue. |
| 08          | xxxx081F    | **Equate Symbol:** ISGENQRsn_NotAuthorizedForExitQName  
**Meaning:** For REQUEST=OBTAIN. An ISGNQXIT exit specified an authorized QNAME for an unauthorized OBTAIN request.  
**Action:** Contact your system programmer. The system programmer should check the ISGNQXIT installation exits to ensure they are not coded to specify an authorized QNAME for unauthorized requests. |
| 08          | xxxx0821    | **Equate Symbol:** ISGENQRsn_NotAuthorizedForOWNINGTOKEN  
**Meaning:** An unauthorized caller specified OWNINGTOKEN.  
**Action:** Unauthorized callers should avoid specifying OWNINGTOKEN. |
| 08          | xxxx0822    | **Equate Symbol:** ISGENQRsn_BadUserDataAddress  
**Meaning:** Unable to access the USERDATA.  
**Action:** Ensure that the entire USERDATA is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Ensure that the storage is in the same key as the caller. |
| 08          | xxxx0823    | **Equate Symbol:** ISGENQRsn_BadUserDataAlet  
**Meaning:** Bad UserData ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller’s Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.  
**Action:** Ensure that the ALET of the userdata is valid. Its access register may not have been set up properly. |
### Table 100. Return and Reason Codes for the ISGENQ Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0824    | Equate Symbol: ISGENQRsn_DeviceNotAllocated  
**Meaning:** For REQUEST=OBTAIN with RESERVEVOLUME=YES. An unauthorized caller specified a device that is not allocated to the requesting task.  
**Action:** Unauthorized callers should allocate the UCB to the job step before ISGENQ RESERVEVOLUME(YES) is issued. |
| 08          | xxxx0825    | Equate Symbol: ISGENQRsn_ExitDeviceNotAllocated  
**Meaning:** For REQUEST=OBTAIN. An ISGNQXIT exit specified a UCB for a device that is not allocated to the requesting, unauthorized task.  
**Action:** Contact your system programmer. The system programmer should ensure that the installation exits do not modify the UCB to specify one that is not allocated to an unauthorized requests. |
| 08          | xxxx0826    | Equate Symbol: ISGENQRsn_BadControl  
**Meaning:** Bad CONTROL keyword parameter.  
**Action:** Check for possible storage overlay of the parameter list. |
| 08          | xxxx0827    | Equate Symbol: ISGENQRsn_BadExitUCB@  
**Meaning:** The storage pointed to by the UCB address changed by a dynamic exit does not map to a valid UCB.  
**Action:** Contact your system programmer. The system programmer should ensure that the installation exits do not specify a bad UCB address. |
| 08          | xxxx0828    | Equate Symbol: ISGENQRsn_NotAuthorizedForENQMAX  
**Meaning:** For REQUEST=OBTAIN, an unauthorized caller specified ENQMAX=NO.  
**Action:** Unauthorized callers should avoid specifying ENQMAX=NO. |
| 0C          | —           | Equate Symbol: ISGENQRc_EnvError  
**Meaning:** ISGENQ request has an environment error.  
**Action:** Refer to action under the individual reason code. |
| 0C          | xxxx0C01    | Equate Symbol: ISGENQRsn_RequestLimitExceeded  
**Meaning:** For REQUEST=OBTAIN, the limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource.  
**Action:** Retry the request one or more times. If the problem persists, consult your system programmer. For more information on concurrent count limits and how the system can be tuned when necessary, see z/OS MVS Planning: Global Resource Serialization. |
### Table 100. Return and Reason Codes for the ISGENQ Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0C          | xxxx0C05    | ISGENQRSn_AbendInExit | **Meaning**: One of the GRS dynamic exits abended.  
**Action**: Retry the request one or more times. Contact your system programmer. |
| 0C          | xxxx0C0A    | ISGENQRSn_TaskEnding  | **Meaning**: The task represented by the specified TToken was ending. The point was reached in task termination after which no ENQs can be obtained.  
**Action**: Determine why the task identified by the TToken was ending. Correct that error and retry the request. |
| 0C          | xxxx0C0B    | ISGENQRSn_FRRHeld  | **Meaning**: The caller issued ISGENQ when an FRR was established.  
**Action**: Avoid issuing ISGENQ when using functional recovery routines. |
| 0C          | xxxx0C0C    | ISGENQRSn_LockHeld | **Meaning**: A lock was held upon entry. No locks can be held when calling ISGENQ.  
**Action**: Avoid using ISGENQ when locks are held. |
| 0C          | xxxx0C0D    | ISGENQRSn_SrbMode  | **Meaning**: ISGENQ was issued while in SRB mode.  
**Action**: Avoid using ISGENQ in SRB mode. |
| 0C          | xxxx0C0E    | ISGENQRSn_NotEnabled | **Meaning**: ISGENQ was issued while not enabled.  
**Action**: Avoid using ISGENQ when not enabled. |
| 0C          | xxxx0C0F    | ISGENQRSn_MasidTarget  | **Meaning**: The requester to be released is still the target of an ENQ with the MASID and MTCB options specified. The release does complete and the resource might be damaged.  
**Action**: The task that issued the ENQ macro instruction with MASID and MTCB should issue the DEQ before this requester does so. |
|             | xxxx0C10    | ISGENQRSn_UnsupportedMode | **Meaning**: The current GRS mode does not support this specific request.  
**Action**: Defer the usage of this particular type of request. |
### Table 100. Return and Reason Codes for the ISGENQ Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
</table>
| 0C          | xxxx0C11    | **Equate Symbol**: ISGENQRsn_MasidNotSupported.  
**Meaning**: The resource that was the target of this REQUEST=CHANGE,CONTROL=SHARED request currently or at one time contained MASID users. REQUEST=CHANGE,CONTROL=SHARED is not supported for resources that involve MASID requestors.  
**Action**: Do not use REQUEST=CHANGE,CONTROL=SHARED on resources that involve MASID requestors. |
| 10          | —           | **Equate Symbol**: ISGENQRc_CompError  
**Meaning**: Component Error.  
**Action**: Contact the IBM Support Center. |
| 10          | xxxx1002    | **Equate Symbol**: ISGENQRsn_CannotObtainHomeStorage  
**Meaning**: ISGENQ processing could not obtain storage in the home address space. |
| 10          | xxxx1003    | **Equate Symbol**: ISGENQRsn_CannotObtainCommonStorage  
**Meaning**: ISGENQ processing could not obtain storage in the common area. |
| 10          | xxxx1004    | **Equate Symbol**: ISGENQRsn_CannotObtainPrimaryAlet  
**Meaning**: ISGENQ processing could not obtain the ALET of the caller's primary address space. |
| 10          | xxxx1006    | **Equate Symbol**: ISGENQRsn_SynchResFlushFailed  
**Meaning**: For REQUEST=OBTAIN, a synchronous reserve failed device state transition flushing. |
| 10          | xxxx1007    | **Equate Symbol**: ISGENQRsn_ReserveStartFailed  
**Meaning**: For REQUEST=OBTAIN, reserve start processing failed. |
| 10          | xxxx1008    | **Equate Symbol**: ISGENQRsn_ReserveCountOverflow  
**Meaning**: For REQUEST=OBTAIN, reserve processing detected an overflow when updating the reserve count. |
| 10          | xxxx1009    | **Equate Symbol**: ISGENQRsn_CannotObtainDSQE  
**Meaning**: ISGENQ processing could not obtain a DSQE to suspend a request during an RNL change. |
| 10          | xxxx100A    | **Equate Symbol**: ISGENQRsn_ReserveDoneFailed  
**Meaning**: For REQUEST=OBTAIN, synchronous reserve back end processing has failed; therefore, the reserve was never completed. |
| 10          | xxxx100B    | **Equate Symbol**: ISGENQRsn_CannotObtainPrimaryStorage  
**Meaning**: ENQ/DEQ processing could not obtain storage in the primary address space. |
Examples

Use these examples as a guide.

* Request exclusive control of a single resource

```
ISGENQ REQUEST=OBTAIN,QNAME=QNAM1,RNAME=RNAM1,RNAMELEN=RLEN1, X
  SCOPE=SYSTEMS,CONTROL=EXCLUSIVE,ENQTOKEN=ENQT1
```

* Release control of a single resource

```
ISGENQ REQUEST=RELEASE,ENQTOKEN=ENQT1,COND=YES,
  RETCODE=(3),RSNCODE=(2)
```

* Conditionally request shared control of 3 resources

```
ISGENQ REQUEST=OBTAIN,RESLIST=YES,NUMRES=3,RESTABLE=RSTBL, X
  ENQTOKEN=ETTBL,RETURN=RTTBL,COND=YES, X
  RETCODE=(3),RSNCODE=(2),PLSTVER=1
```

QNAM1 DC CL8'QNAME1'
RNAM1 DC CL10'RNAME1'
RLEN1 DC AL1(L'RNAM1')
RNAM2 DC CL12'RNAME2'
RNAM3 DC CL14'RNAME3'

DYNAREA DSECT
ENQT1 DS CL(ISGENQENQ_TOKEN_LEN)
ETTBL DS CL(3*ISGENQENQ_TOKEN_LEN)
RTTBL DS CL(3*ISGENQENQ_RETURN_LEN)
ISGENQ REQUEST=OBTAIN,QNAME=QNAM1,RNAME=RNAM1,RNAMELEN=RLEN1, X
SCOPE=SYSTEMS,CONTROL=EXCLUSIVE,ENQTOKEN=ENQT1, X
USERDATA=UDATA1

UDATA1 DC CL32'MY USERDATA'

For more information on global resource serialization, see z/OS MVS Planning: Global Resource Serialization.
Chapter 115. ISGLCRT — Create a latch set

Description

Call the Latch_Create service to create a set of latches. Your application should call Latch_Create during application initialization, and specify a number of latches that is sufficient to serialize all the resources that the application requires. Programs that run as part of the application can call the following related services:

- **ISGLOBT**: Requests exclusive or shared ownership of a latch.
- **ISGLREL**: Releases ownership of an owned latch or a pending request to obtain a latch.
- **ISGLPRG**: Purges all granted and pending requests for a particular requestor within a specific latch set.
- **ISGLID**: Provides a latch set creator the ability to attach a latch identity array to the latch set to identify the individual latches.

In the following description of Latch_Create, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Create.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state or PKM allowing key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be accessible from the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

Before you call the Latch_Create service, include the ISGLMASM macro to obtain assembler declaration statements for Latch_Create. ISGLMASM provides the following equate symbols for use when calling Latch_Create:

* Latch Create Option Equate Symbols
  * ISGLCRT_PRIVATE EQU 0
* Latch Create Return Codes
  *

© Copyright IBM Corp. 1988, 2015
ISGLCRT callable service

ISGLCRT_SUCCESS EQU 0
ISGLCRT_DUPLICATE_NAME EQU 4
ISGLCRT_NO_STORAGE EQU 16
*

Restrictions
You cannot create a latch set in the master scheduler address space if the master scheduler address space is not also the home address space.

Input register information
Before calling the Latch_Create service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 72-byte save area located in the primary address space</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:
- number_of_latches
- latch_set_name
- create_option

Latch_Create returns values in the following parameters:
- latch_set_token
- return_code
**ISGLCRT callable service**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL ISGLCRT</td>
<td>(number_of_latches, latch_set_name, create_option, latch_set_token, return_code)</td>
</tr>
</tbody>
</table>

### Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

- **number_of_latches**
  - Specifies a fullword integer that indicates the number of latches to be created.

- **latch_set_name**
  - Specifies a 48-byte area that contains the name of the latch set. The latch set name must be unique within the current address space. The latch set name can be any value up to 48 characters, but the first character must not be binary zeros or an EBCDIC blank. If the latch set name is less than 48 characters, it must be padded on the right with blanks.

  IBM recommends that you use a standard naming convention for the latch set name. To avoid using a name that IBM uses, do not begin the latch set name with the character string **SYS.** It is a good idea to select a latch set name that is readable in output from the DISPLAY GRS command and interactive problem control system (IPCS). Avoid '@', '$', and '#' because those characters do not always display consistently.

- **create_option**
  - Specifies a fullword integer that must have one of the following values:
    - ISGLCRT_PRIVATE (or a value of 0)
    - ISGLCRT_PRIVATE + ISGLCRT_LOWSTGUSAGE (or a value of 2)
    - ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 (or a value of 64)
    - ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 (or a value of 128)
    - ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 + ISGLCRT_LOWSTGUSAGE (or a value of 66)
    - ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 + ISGLCRT_LOWSTGUSAGE (or a value of 130)

  If the creating address space is constrained by private storage, use the ISGLCRT_LOWSTGUSAGE option. ISGLCRT_LOWSTGUSAGE reduces storage usage at the cost of performance. IBM suggests that this option is only used if there is a known or possible storage constraint issue. See "Specifying the Number of Latches in a Latch Set" in z/OS MVS Programming: Authorized Assembler Services Guide for a description of the amount of storage that can be consumed by a latch set.

  If you want to have the latch obtain services detect some "simple" latch deadlock situations, consider using the ISGLCRT_DEADLOCKDET1 and ISGLCRT_DEADLOCKDET2 options. For performance reasons, latch deadlock detection is not exhaustive. It can detect some simple deadlock situations.
When ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 is specified, it can detect the following deadlock situations:

- The work unit requests exclusive ownership of a latch that the work unit already owns exclusively.
- The work unit requests shared ownership of a latch that the work unit already owns exclusively.

When ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 is specified, it can detect all the deadlock situations listed under ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1, and it can also detect the following situations:

- if the work unit holding a SHARED latch requests exclusive use of the same latch.
- if the work unit holding a SHARED latch requests it SHARED and another unit of work is waiting to obtain the latch EXCLUSIVE.

Because ISGLCRT_DEADLOCKDET2 provides the best deadlock detection, IBM suggests that you use ISGLCRT_DEADLOCKDET1 in cases where it can be used and use ISGLCRT_DEADLOCKDET2 in all cases where there are not many SHARED latch holders.

**Note:**

1. The unit of work context of the requester is captured at latch obtain time. The system does not know if the application passes responsibility for releasing the latch to another unit of work. To prevent false detection, deadlock detection cannot be used if latches are used in such a way that responsibility for releasing the latch is passed between the obtainer and the releaser.
2. Deadlock detection can be safely used by SRBs, if all the obtained latches are released by the SRB work unit before the unit of work completes. There is a possibility of false deadlock hits otherwise.
3. Deadlock detection is not performed if the latches are obtained conditionally using the ISGLOBT_ASYNC_ECB option in ISGLOBT.

**,latch_set_token**

Specifies an 8-byte area to contain the latch set token returned by the Latch_Create service. The latch set token uniquely identifies the latch set. Programs must specify this value on calls to the Latch_Obtain, Latch_Release, and Latch_Purge services.

**,return_code**

A fullword integer to contain the return code from the Latch_Create service.

**ABEND codes**

The caller might encounter abend code X'9C6' for certain errors. See **z/OS MVS System Codes** for explanations and responses.

**Return codes**

When the Latch_Create service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:
### Table 101. ISGLCRT Return Codes

<table>
<thead>
<tr>
<th>Return code and Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (0)</td>
<td><strong>ISGLCRT_SUCCESS</strong>&lt;br&gt;<strong>Meaning:</strong> The Latch_Create service completed successfully.&lt;br&gt;<strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>04 (4)</td>
<td><strong>ISGLCRT_DUPLICATE_NAME</strong>&lt;br&gt;<strong>Meaning:</strong> The specified latch_set_name already exists, and is associated with a latch set that was created by a program running in the current primary address space. The latch manager does not create a new latch set.&lt;br&gt;<strong>Action:</strong> To create a new latch set, specify a unique name on the <code>latch_set_name</code> parameter, then call the <code>Latch_Create</code> service again. Otherwise, continue processing with the returned latch set token.</td>
</tr>
<tr>
<td>10 (16)</td>
<td><strong>ISGLCRT_NO_STORAGE</strong>&lt;br&gt;<strong>Meaning:</strong> Environmental error. Not enough storage was available to contain the requested number of latches. The latch manager does not create a new latch set.&lt;br&gt;<strong>Action:</strong> Specify a smaller value on the <code>number_of_latches</code> parameter.</td>
</tr>
</tbody>
</table>

---

**LATCHX31 - How to call AMODE 31 latch devices**

```plaintext
TITLE 'LATCHX31 - How to call AMODE 31 Latch Services'
*** START OF SPECIFICATIONS **********************************************
* 01* MODULE-NAME = LATCHX31
  * 02* DESCRIPTIVE-NAME = SAMPLE PROGRAM WHICH CONTAINS CALLS
      TO EACH LATCH SERVICE.
  * 01* PROPRIETARY STATEMENT =
  * 01* LICENSED MATERIALS - PROPERTY OF IBM
      THIS MACRO IS "RESTRICTED MATERIALS OF IBM"
  * 01* STATUS = HBB7760
  * 01* FUNCTION:
  *      This module provides samples of how to call the following AMODE 31
      services: ISGLCRT (Latch Create), ISGLID (Latch Id), ISGLOBT
      (Latch Obtain), ISGLREL (Latch Release), ISGLPRG (Latch Purge),
      and ISGLPBA (Latch Purge By Address Space).
  *
  **************************************************************
* 02* RECOVERY-OPERATIONS: None.
*  **************************************************************
* 01* NOTES =
  * 01* (1) Also shows sample of how to allocate an ISGYLID_ENTRY block
      to change the Latch ID field of a latch in the Latchset.
  * 02* DEPENDENCIES: None
  * 02* RESTRICTIONS: None
  * 02* REGISTER-CONVENTIONS:
  * 03* REGISTERS SAVED: R0-R15
*  **************************************************************
```

Chapter 115. ISGLCRT — Create a latch set 1169
**ISGLCRT callable service**

* 03* REGISTERS RESTORED: R2-R14  
  * 03* CODE REGISTER: R12  
  * 03* DATA REGISTER: R13  
  * 02* PATCH-LABEL: None  
  * 01* MODULE-TYPE: Procedure  
  * 02* PROCESSOR: HLASM  
  * 02* MODULE-SIZE: See External Symbol Dictionary  
  * 02* ATTRIBUTES:  
  * 03* LOCATION: User specified  
  * 03* LOAD MODULE: LATCHX31  
  * 03* TYPE: Non-Reentrant  
  * 03* RMODE: Any  
  * 03* SYSGEN: None  
  *  
  ********************************************  
* 01* ENTRY-POINT: LATCHX31  
* 02* PURPOSE: See FUNCTION section for this module.  
* 03* OPERATION: See FUNCTION section for this module.  
* 03* ENTRY  
  * 04* MODE: Enabled  
  * 04* STATE: Problem  
  * 04* KEY: 8  
  * 04* AMODE: 31  
  * 04* LOCKS HELD: None  
  * 04* ASCMODE: Primary  
  * 04* MEMORY MODE: Non-XMEM  
  * 04* DISPATCH MODE: Task  
  * 04* RECOVERY TYPE: None  
  * 04* ADDRESS SPACE: Caller's  
  * 03* EXECUTION  
  * 04* MODE: Enabled  
  * 04* STATE: Supervisor  
  * 04* KEY: 0  
  * 04* AMODE: 31  
  * 04* LOCKS OBTAINED: None  
  * 04* ASCMODE: Primary  
  * 04* MEMORY MODE: Non-XMEM  
  * 04* ADDRESS SPACE: Caller's  
  * 02* LINKAGE: Branched to.  
* 03* CALLERS:  
  *  03* Any  
* 02* INPUT:  
  * 03* ENTRY-REGISTERS:  
  *  03* R0 - R12,R15 - Irrelevant  
  *  ARG-AR15 - Irrelevant  
  *
ISGLCRT callable service

*02* OUTPUT:
*  *
*02* EXIT-NORMAL: RETURN TO CALLER
*  *
*03* CONDITIONS: successful completion
*  *
*03* EXIT-REGISTERS:
*  *
  R0  - R14  - Unchanged
  R15  - Return code (always 0)
  AR0-AR15 - Irrelevant
*  *
*03* RETURN-CODES: None
*  *
* Return code Reason code Interpretation
* ----------- ----------- --------------
*   '0'x    N/A    Success
*  *
*02* EXIT-ERROR: None
*  *
***********************************************************************
*  *
*01* TERMINATION-CONSIDERATIONS: None
*  *
***********************************************************************
*  *
*01* EXTERNAL-REFERENCES:
*  *
*02* ROUTINES: Latch Services.
*  *
*02* DATA-AREAS: None
*  *
*02* CONTROL-BLOCKS:
*  *
  Name   Use Mapping Description
  ------  ----  -----------
  ISGYLID CW  ISGLMASM Latch Identity Entry
*  *
Legend: C=Create, R=Read, W=Write, D=Delete
*  *
*01* TABLES: TRTABLE
*  *
*01* MACROS-EXECUTABLE:
*  *
  None
*  *
*01* SERIALIZATION:
*  *
  None
*  *
*01* MESSAGES:
*  *
  None
*  *
*01* POST-CODES:
*  *
  None
*  *
*01* ABEND-CODES:
*  *
  None
*  *
*01* WAIT-STATE-CODES:
*  *
  None
*  *
ISGLCRT callable service

*01* CHANGE-ACTIVITY:
*  None
*
**** END OF SPECIFICATIONS **********************************************

LATCHX31 CSECT
LATCHX31 AMODE 31
LATCHX31 RMODE ANY

BAKR R14,R0 Save gprs 2-14 and PSW
SAC 0 Ensure primary mode
BRAS R12,PSTART Establish addressability

PSTART EQU *

USING PSTART,12

MODESET MODE=SUP Get into supervisor state
STORAGE OBTAIN,LENGTH=DYNALEN Get savearea and dynamic area
LR R13,R1 Place savearea address into reg13
USING DYNASTORE,R13

MVC 4(4,(R13),=#F1SA' Set the Save area ID (31 bit)

**********************************************************************

* Create latch set
**********************************************************************

CALL ISGLCRT,(NUM_LATCH,LS_NAME,PRIVATE,LS_TOKEN,RETCD), X
MF=(E,CREATE_DPL) –

**********************************************************************

* Initialize Storage for initial LID Entry Block.
* Note that in this example the DYNASTOR section is not freed and
* non-pertinent data is placed in the DYNASTOR section for simplicity.
* The default subpool associated with the STORAGE OBTAIN macro has a
* lifetime of the address space, so the DYNASTOR * section will exist
* for the life of the address space as well.
* However, IBM recommends using a separate storage request
* for the LIDPointerArray and ISGYLID_Entry blocks so that only
* necessary data will continue to exist beyond the life of the calling
* module.
* Also, it is not necessary to initialize all latch IDs in the set
* to point to a default ISGYLID_ENTRY block. NULL values in the
* Latch ID Pointer Array are acceptable.
*
**********************************************************************

* 

XC INIT_STOR,INIT_STOR
XC ENTIRELIDARRAY,ENTIRELIDARRAY
LA R3,INIT_STOR Base ISGYLID_ENTRY block X
at address of allocated storage
LLGTR R3,R3 Clear high half of 64-bit address
USING ISGYLID_ENTRY,R3
LA R4,INITLIDSTR
LLGTR R4,R4 Clear high half of 64-bit address
STG R4,LIDPRINTABLESTRING@
LHI R4,L'INITLIDSTR Put length of INITLIDSTR into X
entry block
STH R4,LIDPRINTABLESTRINGLENGTH
L R4,ONEMINUTE Set hold threshold value to X
one minute
ST R4,LIDHOLDTHRESHOLD
L R4,THIRTYSECONDS Set contention threshold value to X
thirty seconds
ST R4,LIDCONTHRESHOLD

**********************************************************************
Set Latch ID array entries all to address of Initial Lid Entry ---
* indicating all entries are currently unused.
* Latch ID Pointer Array must be in the primary address space.

```
LHI R6,0
L R4,NUM_LATCH
LOOP1 EQU *
STG R3,LIDPTRARRAY(R6)
A R6,LIDPTRARRAYLEN
BCT R4,LOOP1
DROP R3
```

Attach Latch IDs to Latch Set

```
CALL ISGLID,(LS_TOKEN,LIDPTRARRAY,LIDVERSION, X
RETCD),MF=(E,LID_DPL)
```

Change Latch ID Entry for latch #3
* Note:
* Once the Latch Identity Pointer Array has been attached to the
* latch set, it cannot be deleted. However it can be replaced by
* calling the service again and specifying a new array. To change
* the Latch Identity for a particular latch, allocate a new latch
* identity block, fill it out, and update the corresponding array
* entry. Then the program is free to delete the previous latch
* identity block.

```
MVC NEW_STOR,INIT_STOR Copy LID Entry to new storage
USING ISGLID_ENTRY,R3 Use R3 as base for LID Entry Block
LA R3,NEW_STOR
LLGTR R3,R3 Clearing high half of 64-bit address
LA R4,NEWLIDSTR
LLGTR R4,R4 Clearing high half of 64-bit address
STG R4,LIDPRINTABLESTRING@ Store address and length of new X
string in ISGLID_ENTRY block
LHI R5,L'NEWLIDSTR
STH R5,LIDPRINTABLESTRINGLENGTH
L R4,LATCH_NUM Point to new ISGLID_ENTRY block
SLL R4,3 Multiply by 8 (size of LIDPTRARRAY)
STG R3,LIDPTRARRAY(R4)
DROP R3
```

Obtain latch #3

```
MVC REQ_IDH,PSAAOLD-PSA Use ASCB address as high half
MVC REQ_IDL,PSATOLD-PSA Use TCB address as low half
CALL ISGLOBT,(LS_TOKEN,LATCH_NUM,REQ_ID,SUSPEND,EXCLUSIVE, X
ECB_ADDR,LATCH_TOKEN1,WORK_AREA,RETCD), X
MF=(E,OBTAIN_DPL)
```

Release latch #3
ISGLCRT callable service

*******************************************************************************
* CALL ISGLREL,(LS_TOKEN,LATCH_TOKEN1, X UNCOND,WORK_AREA,RETC,DF),MF=(E,RELEASE_DPL)
* Purge requestor from latch set.
* Normally reserved for recovery situations.
*******************************************************************************
* XC REQ_IDL,REQ_IDL Clear the low half of requestor ID
  CALL ISGPLRG,(LS_TOKEN,REQ_ID,RETC),MF=(E,PURGE_DPL)
* Purge all granted and pending requests for a group of requestors for
* a group of latch sets in the current address space.
* Normally reserved for recovery situations.
*******************************************************************************
* CALL ISGPLPA,(=AD(0),REQ_ID,REQ_MASK,LS_NAME,LS_MASK,RETC), X
  MF=(E,PURGEBA_DPL)
* Exit
* Restore caller's regs and return. Also restores caller's PSW key and
* State without a MODESET MODE=PROB.
*******************************************************************************
* LHI R15,0 Set return code to zero
PR ,
* Equates
*******************************************************************************
* R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
* Constants
*******************************************************************************
* NUM_LATCH DC F'16' Number of latches to create - input to create
**ISGLCRT callable service**

```
LS_NAME    DC  CL48'EXAMPLE.ONE_LATCH_SET' latch set name - input to create
LS_MASK    DC  48XL1'FF' Latch set name mask to match all names - input to purge group
REQ_MASK   DC  4XL1'FF'
              DC  4XL1'00' Requestor ID mask to match all requestors with the same first half (ascb0) and any second half
INITLIDSTR DC  C'Latch not used' Initial LID string and length
LIDVERSION DC  AL1(ISGLID_VERSION1) Set latch version to default
LIDPTRARRAYLEN DC  F'8'
THIRTYSECONDS DC  F'30'
ONEMINUTE   DC  F'60'
NEWLIDSTR  DC  C'Serializing Abstract Resource' New string and length
PRIVATE    DC  A(ISGLCRT_PRIVATE) Create option - input to create -
            (defined in IDF)
LATCH_NUM  DC  F'3' Number of latch to be obtained -
            input to obtain
SUSPEND    DC  A(ISGLOBT_SYNC) Obtain option - input to obtain -
            (defined in IDF)
EXCLUSIVE  DC  A(ISGLOBT_EXCLUSIVE) access option - input to obtain -
            (defined in IDF)
UNCOND     DC  A(ISGLREL_UNCOND) Release option - input to release -
            (defined in IDF)
***************************************************************
* Dynamic area for save area, parmlists, and variables
***************************************************************
DYNASTORE  DSECT
SAVEAREA   DS  18F 72-byte register save area
ENTIRELIDARRAY DS 0XL128
LIDPTRARRAY DS 16AD LIDPtrArray
CREATE_DPL  DS  0F
            CALL ,(NUM_LATCH,LS_NAME,PRIVATE,LS_TOKEN,RETCD),MF=L
OBTAIN_DPL  DS  0F
            CALL ,(LS_TOKEN,LATCH_NUM,REQ_ID,SUSPEND,EXCLUSIVE,
                  ECB_ADDR,LATCH_TOKEN1,WORK_AREA,RETCD),MF=L
LID_DPL     DS  0F
            CALL ,(LS_TOKEN,LIDPTRARRAY,LIDVERSION,
                  RETCD),MF=L
RELEASE_DPL DS  0F
            CALL ,(LS_TOKEN,LATCH_TOKEN1,UNCOND,WORK_AREA,RETCD),MF=L
PURGE_DPL   DS  0F
            CALL ,(LS_TOKEN,REQ_ID,RETCD),MF=L
PURGEBA_DPL DS  0F
            CALL ,(0,REQ_ID,REQ_MASK,LS_NAME,LS_MASK,RETCD),MF=L
ECB         DS  F ECB (used only when the obtain
            option is ISGLOBT_ASYNC_ECB)
ECB_ADDR    DS  A Address of ECB - input to obtain -
            (required for the interface, but
            only used when obtain option is
```

Chapter 115. ISGLCRT — Create a latch set 1175
ISGLCRT callable service

* ISGLOBT_ASYNC_ECB
* LS_TOKEN   DS 2F  Latch set token - output from create and input to obtain, latch identity, release, purge, and purge group
* LATCH_TOKEN1 DS 2F  Latch token - output from obtain and input to release
* REQ_ID     DS 8D  Double word requestor ID - input to obtain, purge, and purge group
* REQ_IDH    DS F  First half of requestor ID (ascb@)
* REQ_IDL    DS F  Second half of requestor ID (tcb@)
* WORK_AREA  DS 32D 256-byte work area
* RETCD      DS F  Return code - output from services
* INIT_STOR  DS XL(ISGLID_ENTRY_LEN) Isglid_entry block storage
* NEW_STOR   DS XL(ISGLID_ENTRY_LEN)
* DYNALEN    EQU *-DYNASTORE
* IHAPSA     , Needed for ascb@ and tcb@ req ID
* ISGLMASM   , Needed for latch constants
END LATCHX31
Chapter 116. ISGLID — Identify a latch set

Description

The ISGLID callable service provides a latch set creator the ability to attach a latch identity array (in ISGLMASM or ISGLMC) to the latch set for the purposes of identifying the individual latches in the latch set. The LIDArray must be in the primary space. The following callable services are related to the ISGLID service:

- ISGLCRT
- ISGLCRT64
- ISGLID
- ISGLID64

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PKM allowing key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Primary = the space of the latch set creator</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be accessible from the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

After the latch identity pointer array has been attached to the latch set, the attached LIDArray cannot be deleted. However, the LIDArray can be replaced by calling the service again and specifying a new array. To change the latch identity for a particular latch, allocate a new latch identity block, fill it out, and update the corresponding array entry. Then, you can delete the previous latch identity block.

Restrictions

None.

Input register information

Before calling the ISGLID service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 72-byte save area located in the primary address space.</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:
ISGLID callable service

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1 Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13 Unchanged.</td>
</tr>
<tr>
<td>14-15 Used as work registers by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1 Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13 Unchanged.</td>
</tr>
<tr>
<td>14-15 Used as work registers by the system.</td>
</tr>
</tbody>
</table>

Performance implications

None

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- lsetToken
- LIDPtrArray
- LIDEntryVersion

ISGLID returns values in the following parameter:

- retcode

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL ISGLID</td>
<td>(lsetToken, LIDPtrArray, LIDEntryVersion, retcode)</td>
</tr>
</tbody>
</table>

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

- lsetToken
  Specifies an 8-character field that contains the latch set token returned from ISGLCRT.

- LIDPtrArray
  Specifies the latch identity pointer array.

- LIDEntryVersion
  Specifies a 1-byte area that contains the version of the LID entries.
RetCode
Specifies a 4-byte or 32-bit area that contains return code from the ISGLID service.

ABEND codes
The caller might encounter abend code X'9C6' for certain errors. See z/OS MVS System Codes for explanations and responses for these codes.

Return codes
When the ISGLID service returns control to your program, the RetCode parameter contains a hexadecimal return code. The following table identifies the hexadecimal reason codes and meaning associated with each reason code:

Table 102. ISGLID Return Codes

<table>
<thead>
<tr>
<th>Return code and Equate symbol</th>
<th>Meaning and action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 ISGLID_SUCCESS</td>
<td><strong>Meaning:</strong> The ISGLID service completed successfully.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> None.</td>
</tr>
<tr>
<td>xxxx0401 ISGLID_REPLACED</td>
<td><strong>Meaning:</strong> Latch identity pointer array is replaced. A previous latch identity pointer array existed for this latch set. It has been replaced.</td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> None.</td>
</tr>
</tbody>
</table>

Example
See “LATCHX31 - How to call AMODE 31 latch devices” on page 1169.
Chapter 117. ISGLOBT — Obtain a latch

Description

Call the Latch_Obtain service to request exclusive or shared ownership of a latch. When a requestor owns a particular latch, the requestor can use the resource associated with that latch. The following callable services are related to Latch_Obtain:

**ISGLCRT**
Creating a latch set that an application can use to serialize resources.

**ISGLREL**
Releases ownership of an owned latch or a pending request to obtain a latch.

**ISGLPRG**
Purges all granted and pending requests for a particular requestor within a specific latch set.

In the following description of Latch_Obtain:

- The term requestor describes a task or SRB routine that calls the Latch_Obtain service to request ownership of a latch.
- Equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Obtain. For example, "ISGLOBT_COND (value of 1)" indicates the equate symbol ISGLOBT_COND and its associated value, 1.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PKM allowing key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be accessible from the primary address space.</td>
</tr>
</tbody>
</table>

Programming Requirements

If you specify an obtain option of ISGLOBT_ASYNC_ECB (as described in "Parameters on page 1183"), initialize the ECB pointed to by the value on the ECB_address parameter to zero before calling Latch_Obtain.

Before you use the Latch_Obtain service, you need to include the ISGLMASM macro to obtain assembler declaration statements for Latch_Obtain. ISGLMASM provides the following equate symbols for use when calling Latch_Obtain:

- Latch Obtain Option Equate Symbols

© Copyright IBM Corp. 1988, 2015
ISGLOBT callable service

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISGLOBT_SYNC</td>
<td>0</td>
</tr>
<tr>
<td>ISGLOBT_COND</td>
<td>1</td>
</tr>
<tr>
<td>ISGLOBT_ASYNC_ECB</td>
<td>2</td>
</tr>
<tr>
<td>ISGLOBT_EXCLUSIVE</td>
<td>0</td>
</tr>
<tr>
<td>ISGLOBT_SHARED</td>
<td>1</td>
</tr>
<tr>
<td>ISGLOBT_SUCCESS</td>
<td>0</td>
</tr>
<tr>
<td>ISGLOBTCONTENTION</td>
<td>4</td>
</tr>
</tbody>
</table>

Restrictions

1. The caller of Latch_Obtain must have a PSW key that allows access to the latch set storage.
2. The ECB specified on the ECB_address parameter must reside in storage with a storage key that matches the latch set storage key.
3. You must call Latch_Obtain from the same primary address space from which the Latch_Create service was called.

Input register information

Before calling the Latch_Obtain service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 72-byte save area located in the primary address space</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications

See the information about obtaining latches in z/OS MVS Programming: Authorized Assembler Services Guide for performance implications related to the Latch_Obtain service.
Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:
- latch_set_token
- latch_number
- requestor_ID
- obtain_option
- access_option
- ECB_address

Latch_Obtain returns values in the following parameters:
- latch_token
- return_code

Latch_Obtain uses the following parameter for temporary storage:
- work_area

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL ISGLOBT</td>
<td>(latch_set_token, latch_number, requestor_ID, obtain_option, access_option, ECB_address, latch_token, work_area, return_code)</td>
</tr>
</tbody>
</table>

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

**latch_set_token**
Specifies an 8-byte area that contains the latch_set_token that the Latch_Create service returned earlier when it created the latch set.

**latch_number**
Specifies a fullword integer that contains the number of the latch to be obtained. The latch_number must be in the range from 0 to the total number of latches in the associated latch set minus one.

**requestor_ID**
Specifies an 8-byte area that contains a value that identifies the caller of the Latch_Obtain service. The requestor_ID can be any value except all binary zeros.

Recovery routines can purge all granted and pending requests for a particular requestor (identified by a requestor_id) within a specific latch set. When specifying the requestor_ID on Latch_Obtain, consider which latches would be purged if the Latch_Purge service were to be called with the specified...
requestor_ID. For more information about the Latch_Purge service, see Chapter 119, “ISGPRG — Purge a requestor from a latch set,” on page 1193.

,obtain_option
A fullword integer that specifies how the system is to handle the Latch_Obtain request if the latch manager cannot immediately grant ownership of the latch to the requestor:

**ISGLOBT_SYNC (value of 0)**
The system processes the request synchronously. The system suspends the requestor. When the latch manager eventually grants ownership of the latch to the requestor, the system returns control to the requestor.

**ISGLOBTCOND (value of 1)**
The system processes the request conditionally. The system returns control to the requestor with a return code of ISGLOBT_CONTENTION (value of 4). The latch manager does not queue the request to obtain the latch.

**ISGLOBT_ASYNC_ECB (value of 2)**
The system processes the request asynchronously. The system returns control to the requestor with a return code of ISGLOBT_CONTENTION (value of 4). When the latch manager eventually grants ownership of the latch to the requestor, the system posts the ECB pointed to by the value specified on the ECB_address parameter.

When you specify this option, the ECB_address parameter must contain the address of an initialized ECB that is addressable from the home address space (HASN).

,access_option
A fullword or character string that specifies the access required:

- **ISGLOBT_EXCLUSIVE (value of 0)** - Exclusive (write) access
- **ISGLOBT_SHARED (value of 1)** - Shared (read) access

,ECB_address
Specifies a fullword that contains the address of an ECB. If you specify an obtain_option of ISGLOBT_SYNC (value of 0) or ISGLOBT_COND (value of 1) on the call to Latch_Obtain, the ECB_address field must be valid (though its contents are ignored). IBM recommends that an address of 0 be used when no ECB is to be processed.

If you specify an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) and the system returns a return code of ISGLOBT_CONTENTION (value of 4) to the caller, the system posts the ECB pointed to by the value specified on the ECB_address parameter when the latch manager grants ownership of the latch to the requestor.

,latch_token
Specifies an 8-byte area to contain the latch token returned by the Latch_Obtain service. You must provide this value as a parameter on a call to the Latch_Release service to release the latch.

,work_area
Specifies a 256-byte work area that provides temporary storage for the Latch_Obtain service. The work area should begin on a doubleword boundary to optimize performance. The work area must be in the same storage key as the caller of Latch_Obtain.

,return_code
Specifies a fullword integer that is to contain the return code from the Latch_Obtain service.
ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See z/OS MVS System Codes for explanations and responses for these codes.

Return codes

When the Latch_Obtain service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 103. ISGLOBT Return Codes

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0) ISGLOBT_SUCCESS</td>
<td>Meaning: The Latch_Obtain service completed successfully.</td>
</tr>
<tr>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>04 (4) ISGLOBT_CONTENTION</td>
<td>Meaning: A requestor called Latch_Obtain with an obtain_option of ISGLOBT_COND (value of 1) or ISGLOBT_ASYNC_ECB (value of 2). The latch is not immediately available.</td>
</tr>
<tr>
<td></td>
<td>Action: If the requestor specified an obtain_option of ISGLOBT_COND (value of 1), no response is required. If the requestor specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2), and the latch is still required, wait on the ECB to be posted when the latch manager grants ownership of the latch to the requestor.</td>
</tr>
</tbody>
</table>

Example

See “LATCHX31 - How to call AMODE 31 latch devices” on page 1169 for an example of how to call Latch_Obtain in assembler language.
Chapter 118. ISGLREL — Release a latch

Description

Call the Latch_Release service to release ownership of an owned latch or a pending request to obtain a latch. Requestors should call Latch_Release when the use of a resource associated with a latch is no longer required. The following callable services are related to Latch_Release:

ISGLCRT
    Creates a latch set that an application can use to serialize resources.

ISGLOBT
    Requests exclusive or shared control of a latch.

ISGLPRG
    Purges all granted and pending requests for a particular requestor within a specific latch set.

In the following description of Latch_Release:

- The term requestor describes a program that calls the Latch_Release service to release ownership of an owned latch or a pending request to obtain a latch.
- Equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Obtain. For example, "ISGLREL_COND (value of 1)" indicates the equate symbol ISGLREL_COND and its associated value, 1.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PKM allowing key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be accessible from the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

Before you use the Latch_Release service, include the ISGLMASM macro to obtain assembler declaration statements for Latch_Release. ISGLMASM provides the following equate symbols for use when calling Latch_Release:

* Latch Release Option Equate Symbols
  * ISGLREL_UNCOND EQU 0
  * ISGLREL_COND EQU 1

* Latch Release Return Codes
  * ISGLREL_SUCCESS EQU 0
ISGLREL callable service

ISGLREL_NOT_OWNED_ECB_REQUEST EQU 4
ISGLREL_STILL_SUSPENDED EQU 8
ISGLREL_INCORRECT_LATCH_TOKEN EQU 12

Restrictions
1. The caller of Latch_Release must have a PSW key that allows access to the latch set storage.
2. You must call Latch_Release from the same primary address space from which the Latch_Create service was called.

Input register information
Before calling the Latch_Release service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register      Contents
13            Address of a standard 72-byte save area located in the primary address space

Output register information
When control returns to the caller, the GPRs contain:

Register      Contents
0-1           Used as work registers by the system
2-13          Unchanged
14-15         Used as work registers by the system

When control returns to the caller, the ARs contain:

Register      Contents
0-1           Used as work registers by the system
2-13          Unchanged
14-15         Used as work registers by the system

Performance implications
See the information about releasing latches in z/OS MVS Programming: Authorized Assembler Services Guide for performance implications related to the Latch_Release service.

Syntax
Write the CALL as shown on the syntax diagram, coding all parameters in the specified order.

Assign values to the following parameters:
- latch_set_token
- latch_token
- release_option
ISGLREL callable service

Latch_Release returns a value in the following parameter:
- return_code

Latch_Release uses the following parameter for temporary storage:
- work_area

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL ISGLREL</td>
<td>,(latch_set_token, latch_token, release_option, work_area, return_code)</td>
</tr>
</tbody>
</table>

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

*latch_set_token*
   Specifies an 8-byte area that contains the latch set token returned to the caller of the Latch_Create service. The latch set token identifies the latch set that contains the latch to be released.

*latch_token*
   Specifies an 8-byte area that contains the latch token returned to the caller of the Latch_Obtain service. The latch token identifies the request to be released.

*release_option*
   Specifies a fullword integer that tells the latch manager what to do when the requestor either no longer owns the latch to be released or still has a pending request to obtain the latch to be released:

**ISGLREL_UNCOND (value of 0)**
   Abend the requestor:
   - If a requestor originally specified an obtain_option of ISGLOBT_SYNC (value of 0) and is suspended while waiting to obtain the latch, the latch manager does not release the latch. The system abends the caller of Latch_Release with abend 'X9C6', reason code xxxx0009.
   - If a requestor originally specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) and is suspended while waiting to obtain the latch, the latch manager does not release the latch. The system abends the caller of Latch_Release with abend 'X9C6', reason code xxxx0007.
   - If the latch manager does not find a previous Latch_Obtain request for the specified latch, the system abends the caller of Latch_Release with abend 'X9C6', reason code xxxx000A.

**ISGLREL_COND (value of 1)**
   Return control to the requestor:
   - If a requestor originally specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) and the latch has been obtained, but the ECB has not been posted, the latch manager releases the request for ownership of the latch. The system returns control to the caller of
ISGLREL callable service

Latch_Release with a return code of ISGLREL_NOT_OWNED_ECB_REQUEST (value of 4).

- If a requestor originally specified an obtain_option of ISGLOBT_SYNC (value of 0) but is suspended while waiting to obtain the latch, the latch manager does not release the request for ownership of the latch. The system returns control to the caller of Latch_Release with a return code of ISGLREL_STILL_SUSPENDED (value of 8).
- If the latch manager does not find a previous Latch_Obtain request for the specified latch, the system returns control to the caller of Latch_Release with a return code of ISGLREL_INCORRECT_LATCH_TOKEN (value of 12).

,work_area

Specifies a 256-byte work area that provides temporary storage for the Latch_Release service. The work area should begin on a doubleword boundary to optimize performance. The work area must be in the same storage key as the caller of Latch_Release.

$return_code

Specifies a fullword integer that is to contain the return code from the Latch_Release service.

ABEND codes

The caller might encounter abend code X’9C6’ for certain errors. See z/OS MVS System Codes for explanations and responses.

Return codes

When the Latch_Release service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 104. ISGLREL Return Codes

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00 (0) ISGLREL_SUCCESS | **Meaning:** The Latch_Release service completed successfully. The caller released ownership of the specified latch request.  
**Action:** None. |
| 04 (4) ISGLREL_NOT_OWNED_ECB_REQUEST | **Meaning:** The requestor that originally called the Latch_Obtain service is still expecting the system to post an ECB (to indicate that the requestor has obtained the latch). The call to the Latch_Release service specified a release_option of ISGLREL_COND (value of 1). The latch manager does not post the ECB at the address specified on the original call to Latch_Obtain. The latch manager releases the latch.  
**Action:** Validate the integrity of the resource associated with the latch (the requestor might have used the resource without waiting on the ECB). If the resource is undamaged, no action is necessary (a requestor routine may have been in the process of cancelling the request to obtain the latch). |
### ISGLREL callable service

**Table 104. ISGLREL Return Codes (continued)**

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08 (8) ISGLREL_STILL_SUSPENDED | **Meaning**: Program error. The request specified a correct latch token, but the program that originally requested the latch is still suspended and waiting to obtain the latch.  
The latch requestor originally specified an obtain_option of ISGLOBT_SYNC on the call to the Latch_Obtain service. The call to the Latch_Release service specified a release_option of ISGLREL_COND (value of 1). The latch manager does not release the latch. The latch requestor remains suspended.  
**Action**:  
• Wait for the latch requestor to obtain the latch and receive control back from the system; then call the Latch_Release service again, or  
• End the program that originally requested the latch. |
| 0C (12) ISGLREL_INCORRECT_LATCH_TOKEN | **Meaning**: The latch manager could not find a granted or pending request associated with the value on the latch token parameter. The latch manager does not release a latch.  
This return code does not indicate an error if a routine calls Latch_Release to ensure that a latch is released. For example, if an error occurs when a requestor calls the Latch_Obtain service, the requestor’s recovery routine might call Latch_Release to ensure that the requested latch is released. If the error prevented the requestor from obtaining the latch, the recovery routine receives this return code.  
**Action**: If the return code is not expected, validate that the latch token is the same latch token returned to the caller of Latch_Obtain. |

### Example

See [“LATCHX31 - How to call AMODE 31 latch devices” on page 1169](#) for an example of how to call Latch_Release in assembler language.
Chapter 119. ISGLPRG — Purge a requestor from a latch set

Description

Call the Latch_Purge service to purge all granted and pending requests for a particular requestor within a specific latch set. Recovery routines should call Latch_Purge when one or more errors prevent requestors from releasing latches. The following callable services are related to Latch_Purge:

ISGLCRT
Creates a latch set that an application can use to serialize resources.

ISGLOBT
Requests exclusive or shared control of a latch.

ISGLREL
Releases control of an owned latch or a pending request to obtain a latch.

In the following description of Latch_Purge, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Purge.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PKM allowing key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be accessible from the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

Before you use the Latch_Purge service, you need to include the ISGLMASM macro to obtain assembler declaration statements for Latch_Purge. ISGLMASM provides the following equate symbols for use when calling Latch_Purge:

* Latch Purge Return Codes

* ISGLPRG_SUCCESS EQU 0
* ISGLPRG_DAMAGE_DETECTED EQU 4

Restrictions

1. The caller of Latch_Purge must have a PSW key that allows access to the latch set storage.
2. You must call Latch_Purge from the same primary address space from which the Latch_Create service was called.
ISGLPRG callable service

Input register information
Before calling the Latch_Purge service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 72-byte save area located in the primary address space</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications
None.

Syntax
Write the CALL as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:
- latch_set_token
- requestor_ID

Latch_Purge returns a value in the return_code parameter.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL ISGLPRG</td>
<td>./latch_set_token .requestor_ID .return_code</td>
</tr>
</tbody>
</table>

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:
ISGLPRG callable service

latch_set_token
 Specifies an 8-byte area that contains the latch_set_token previously returned by the Latch_Create service. The latch set token identifies the latch set from which latch requests are to be purged.

requestor_ID
 Specifies an 8-byte area that contains the requestor_ID originally specified on one or more previous calls to the Latch_Obtain service. The Latch_Purge service is to release all Latch_Obtain requests that specify this requestor_ID.

return_code
 A fullword integer that contains the return code from the Latch_Purge service.

ABEND codes
 The caller might encounter abend code X’9C6’ for certain errors. See z/OS MVS System Codes for explanations and responses.

Return codes
 When the Latch_Purge service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (0) ISGLPRG_SUCCESS</td>
<td>Meaning: The Latch_Purge service completed successfully. Action: None.</td>
</tr>
<tr>
<td>04 (4) ISGLPRG_DAMAGE_DETECTED</td>
<td>Meaning: Program error. While purging all requests for a particular requestor from a latch set, the latch manager found incorrect data in one or more latches. The latch manager tries to purge the latches that contain incorrect data, but the damage might prevent the latch manager from purging those latches. The latch manager purges the remaining latches (those with correct data) for the specified requestor. Action: Take a dump and check for a storage overlay. If your application can continue without the resources serialized by the damaged latches, no action is required.</td>
</tr>
</tbody>
</table>

Example
 See "LATCHX31 - How to call AMODE 31 latch devices" on page 1169 for an example of how to call Latch_Purge in assembler language.
Chapter 120. ISGLPBA — Purge a group of requestors from a group of latch sets

Description

Call the Latch_Purge_by_Address_Space service to purge all granted and pending requests for a group of requestors for a group of latch sets in the same address space. To effectively use this service, your latch_set_names and your requestor_IDs should be defined such that they have a common portion and a unique portion. Groups of latch sets can then be formed by masking off the unique portion of the latch_set_name, and groups of latch requests in a latch set can then be formed by masking off the unique portion of the requestor_ID. Masking off the unique portion of the requestor_ID allows a single purge request to handle multiple latch sets and multiple requests in a latch set. Recovery routines should call Latch_Purge_by_Address_Space when one or more errors prevent requestors from releasing latches.

The following callable services are related to Latch_Purge_by_Address_Space:

**ISGLCRT**

Creates a latch set that an application can use to serialize resources.

**ISGLOBT**

Requests exclusive or shared control of a latch.

**ISGLREL**

Releases control of an owned latch or a pending request to obtain a latch.

**ISGLPRG**

Purges all granted and pending requests for a particular requestor within a specific latch set.

In the following description of Latch_Purge_by_Address_Space, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Purge_by_Address_Space.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state or PKM allowing key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be accessible from the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

Before you use the Latch_Purge_by_Address_Space service, you need to include the ISGLMASM macro to obtain assembler declaration statements for
ISGLPBA callable service

Latch_Purge_by_Address_Space. ISGLMASM provides the following equate symbols for use when calling Latch_Purge_by_Address_Space

* *
* Latch Purge by Address Space Return Codes
* *
ISGLPRG_SUCCESS EQU 0
ISGLPRG_DAMAGE_DETECTED EQU 4
ISGLPRG_INCORRECT_MASK EQU C
* *

Restrictions
1. The caller of Latch_Purge_by_Address_Space must have a PSW key that allows access to the latch set storage.
2. You must call Latch_Purge_by_Address_Space from the same primary address space from which the Latch_Create service was called.

Input register information
Before calling the Latch_Purge_by_Address_Space service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 72-byte save area located in the primary address space</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications
None.

Syntax
Write the CALL as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:
- latch_set_token
ISGLPBA callable service

- requestor_ID
- requestor_ID_mask
- latch_set_name
- latch_set_name_mask

Latch_Purge_by_Address_Space returns a value in the return_code parameter.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL ISGLPBA</td>
<td>(latch_set_token, requestor_ID, requestor_ID_mask, latch_set_name, latch_set_name_mask, return_code)</td>
</tr>
</tbody>
</table>

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

- **latch_set_token**
  Specifies an 8-byte area that contains the latch_set_token previously returned by the Latch_Create service or a value of zero. If the value is not zero, the latch_set_token identifies the latch set from which latch requests are to be purged. If the latch_set_token is set to zero, a group of latch sets, determined by the latch_set_name and latch_set_name_mask, will have their latch requests purged.

- **requestor_id**
  Specifies an 8-byte area that contains a portion of the requestor_ID originally specified on one or more previous calls to the Latch_Obtain service. This operand will be compared to the result of logically ANDing each requestor_ID in the latch set with the requestor_ID_mask. Make sure that any corresponding bits that are zero in the requestor_ID_mask are also zero in this field, otherwise no ID matches will occur. Each requestor_ID that has a name match will have its Latch_Obtain requests released.

- **requestor_id_mask**
  Specifies an 8-byte area that contains the requestor_ID_mask that will be logically ANDed to each requestor_ID in the latch set and then compared to the requestor_ID operand. Each requestor_ID that has a name match will have its Latch_Obtain requests released.

- **latch_set_name**
  Specifies a 48-byte area that contains the portion of the latch_set_name that will be compared to the result of logically ANDing the latch_set_name_mask with each latch set name in the primary address space. Make sure that any corresponding bits that are zero in the latch_set_name_mask are also zero in this field, otherwise no name matches will occur. Each latch set that has a name match will have its Latch_Obtain requests released. If the latch_set_token operand is non-zero this operand is ignored.

- **latch_set_name_mask**
  Specifies a 48-byte area that contains the mask that will be logically ANDed to each of the latch set names in the primary address space and then compared to
**ISGLPBA callable service**

the latch_set_name operand. Each latch set that has a name match will have its Latch_Obtain requests released. If the latch_set_token operand is non-zero this operand is ignored.

**Return_code**

A fullword integer that contains the return code from the Latch_Purge_By_Address_Space service.

**ABEND codes**

The caller might encounter abend code X'9C6' for certain errors. See z/OS MVS System Codes for explanations and responses.

**Return codes**

When the Latch_Purge_by_Address_Space service returns control to your program, the return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

*Table 106. ISGLPBA Return Codes*

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (0) ISGLPRG_SUCCESS</td>
<td><strong>Meaning:</strong> The Latch_Purge_by_Address_Space service completed successfully. <strong>Action:</strong> None.</td>
</tr>
<tr>
<td>04 (4) ISGLPRG_DAMAGE_DETECTED</td>
<td><strong>Meaning:</strong> Program error. While purging all requests for a particular requestor from a latch set, the latch manager found incorrect data in one or more latches. The latch manager tries to purge the latches that contain incorrect data, but the damage might prevent the latch manager from purging those latches. The latch manager purges the remaining latches (those with correct data) for the specified requestor. <strong>Action:</strong> Take a dump and check for a storage overlay. If your application can continue without the resources serialized by the damaged latches, no action is required.</td>
</tr>
</tbody>
</table>
Chapter 121. ISGLCR64 — Create a latch set in 64-bit mode

Description
Call the 64-bit Latch_Create service to create a set of latches. Your application should call Latch_Create during application initialization, and specify a number of latches that is sufficient to serialize all the resources that the application requires. Programs that run as part of the application can call the following related services:

ISGLOB64
Requests exclusive or shared ownership of a latch.

ISGLRE64
Releases ownership of an owned latch or a pending request to obtain a latch.

ISGLPR64
Purges all granted and pending requests for a particular requestor within a specific latch set.

ISGLID64
Provides a latch set creator the ability to attach a latch identity array to the latch set to identify the individual latches.

In the following description of 64-bit Latch_Create, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Create.

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PSW allowing key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be accessible from the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements
Before you call the 64-bit Latch_Create service, include the ISGLMASM macro to obtain assembler declaration statements for 64-bit Latch_Create. ISGLMASM provides the following equate symbols for use when calling Latch_Create:

* Latch Create Option Equate Symbols
  * ISGLCRT_PRIVATE EQU 0
* Latch Create Return Codes
  *
ISGLCR64 callable service

ISGLCRT_SUCCESS EQU 0
ISGLCRT_DUPLICATE_NAME EQU 4
ISGLCRT_NO_STORAGE EQU 16

* Restrictions

You cannot create a latch set in the master scheduler address space if the master scheduler address space is not also the home address space.

Input register information

Before calling the 64-bit Latch_Create service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 144-byte save area located in the primary address space</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- number_of_latches
- latch_set_name
- create_option

The 64-bit Latch_Create returns values in the following parameters:

- latch_set_token
- return_code
CALL ISGLCR64(number_of_latches,latch_set_name,create_option,latch_set_token,return_code)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

**number_of_latches**
Specifies a fullword integer that indicates the number of latches to be created.

**latch_set_name**
Specifies a 48-byte area that contains the name of the latch set. The latch set name must be unique within the current address space. The latch set name can be any value up to 48 characters, but the first character must not be binary zeros or an EBCDIC blank. If the latch set name is less than 48 characters, it must be padded on the right with blanks.

IBM recommends that you use a standard naming convention for the latch set name. To avoid using a name that IBM uses, do not begin the latch set name with the character string **SYS**. It is a good idea to select a latch set name that is readable in output from the DISPLAY GRS command and interactive problem control system (IPCS). Avoid '@', '$', and '#' because those characters do not always display consistently.

**create_option**
Specifies a fullword integer that must have one of the following values:

- ISGLCRT_PRIVATE (or a value of 0)
- ISGLCRT_PRIVATE + ISGLCRT_LOWSTGUSAGE (or a value of 2)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 (or a value of 64)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 (or a value of 128)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 + ISGLCRT_LOWSTGUSAGE (or a value of 66)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 + ISGLCRT_LOWSTGUSAGE (or a value of 130)

If the creating address space is constrained by private storage, use the ISGLCRT_LOWSTGUSAGE option. ISGLCRT_LOWSTGUSAGE reduces storage usage at the cost of performance. IBM suggests that this option is only used if there is a known or possible storage constraint issue. See "Specifying the Number of Latches in a Latch Set" in z/OS MVS Programming: Authorized Assembler Services Guide for a description of the amount of storage that can be consumed by a latch set.

If you want to have the latch obtain services detect some "simple" latch deadlock situations, consider using the ISGLCRT_DEADLOCKDET1 and ISGLCRT_DEADLOCKDET2 options. For performance reasons, latch deadlock detection is not exhaustive. It can detect some simple deadlock situations.
When ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 is specified, it can detect the following deadlock situations:

- The work unit requests exclusive ownership of a latch that the work unit already owns exclusively.
- The work unit requests shared ownership of a latch that the work unit already owns exclusively.

When ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 is specified, it can detect all the deadlock situations listed under ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1, and it can also detect if the work unit holding a SHARED latch requests exclusive use of the same latch. It also catches multiple Share requests by same unit of work when there is exclusive Waiter in between -> Shared (UW1) - Exclusive (UW2) - Shared (UW1).

Because ISGLCRT_DEADLOCKDET2 provides the best deadlock detection, IBM suggests that you use ISGLCRT_DEADLOCKDET1 in cases where it can be used and use ISGLCRT_DEADLOCKDET2 in all cases where there are not many SHARED latch holders.

**Note:**

1. The unit of work context of the requester is captured at latch obtain time. The system does not know if the application passes responsibility for releasing the latch to another unit of work. To prevent false detection, deadlock detection cannot be used if latches are used in such a way that responsibility for releasing the latch is passed between the obtainer and the releaser.

2. Deadlock detection can be safely used by SRBs, if all the obtained latches are released by the SRB work unit before the unit of work completes. There is a possibility of false deadlock hits otherwise.

3. Deadlock detection is not performed if the latches are obtained conditionally using the ISGLOBT_ASYNC_ECB option in ISGLOBT.

,,latch_set_token

Specifies an 8-byte area to contain the latch set token returned by the Latch_Create service. The latch set token uniquely identifies the latch set. Programs must specify this value on calls to the Latch_Obtain, Latch_Release, and Latch_Purge services.

,,return_code

A fullword integer to contain the return code from the Latch_Create service.

**ABEND codes**

The caller might encounter abend code X'9C6' for certain errors. See z/OS MVS System Codes for explanations and responses.

**Return codes**

When the Latch_Create service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

<table>
<thead>
<tr>
<th>Return code and Equate symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (0) ISGLCRT_SUCCESS</td>
<td>Meaning: The Latch_Create service completed successfully.</td>
</tr>
<tr>
<td></td>
<td>Action: None required.</td>
</tr>
</tbody>
</table>
Table 107. ISGLCR64 Return Codes (continued)

<table>
<thead>
<tr>
<th>Return code and Equate symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>04 (4) ISGLCRT_DUPLICATE_NAME</td>
<td>The specified latch_set_name already exists, and is associated with a latch set that was created by a program running in the current primary address space. The latch manager does not create a new latch.</td>
<td>To create a new latch set, specify a unique name on the latch_set_name parameter, then call the Latch_Create service again. Otherwise, continue processing with the returned latch set token.</td>
</tr>
<tr>
<td>10 (16) ISGLCRT_NO_STORAGE</td>
<td>Environmental error. Not enough storage was available to contain the requested number of latches. The latch manager does not create a new latch set.</td>
<td>Specify a smaller value on the number_of_latches parameter.</td>
</tr>
</tbody>
</table>

LATCHX64 - How to call AMODE 64 latch services

Title 'LATCHX64 - How to call AMODE 64 Latch Services'

* *** START OF SPECIFICATIONS ***************************************************************
* +01* MODULE-NAME = LATCHX64
* +02* DESCRIPTIVE-NAME = SAMPLE PROGRAM WHICH Contains CALLS
* To EACH LATCH SERVICE.
* +01* PROPRIETARY STATEMENT =
* + LICENSED MATERIALS - PROPERTY OF IBM
* + THIS MACRO IS "RESTRICTED MATERIALS OF IBM"
* +01* STATUS = HBB7760
* +01* FUNCTION:
* + This module provides samples of how to call the following AMODE 64
* + Services: ISGLCR64 (Latch Create), ISGLID64 (Latch Id), ISGLOB64
* + (Latch Obtain), ISGLRE64 (Latch Release), ISGLPR64 (Latch Purge),
* + and ISGLPB64 (Latch Purge By Address Space).
* +***********************************************************************
* +02* RECOVERY-OPERATIONS: None.
* +***********************************************************************
* +01* NOTES =
* + (1) Also shows sample of how to allocate an ISGYLID_ENTRY block
* + to change the Latch ID field of a latch in the Latchset.
* +02* DEPENDENCIES: None
* +02* RESTRICTIONS: None
* +02* REGISTER-CONVENTIONS:
* +03* REGISTERS SAVED: R0-R15
* +03* REGISTERS RESTORED: R2-R14
* +03* CODE REGISTER: R12

ISGLCR64 callable service

Chapter 121. ISGLCR64 — Create a latch set in 64-bit mode 1205
**ISGLCR64 callable service**

```
* 03* DATA REGISTER: R13
* 02* PATCH-LABEL: None
* 01* MODULE-TYPE: Procedure
* 02* PROCESSOR: HLASM
* 02* MODULE-SIZE: See External Symbol Dictionary
* 02* ATTRIBUTES:
  * 03* LOCATION: User specified
  * 03* LOAD MODULE: LATCHX64
  * 03* TYPE: Non-Reentrant
  * 03* RMODE: Any
  * 03* SYSGEN: None

***********************************************************************
* 01* ENTRY-POINT: LATCHX64
* 02* PURPOSE: See FUNCTION section for this module.
* 03* OPERATION: See FUNCTION section for this module.
* 03* ENTRY
  * 04* MODE: Enabled
  * 04* STATE: Problem
  * 04* KEY: 8
  * 04* AMODE: 64
  * 04* LOCKS HELD: None
  * 04* ASCMODE: Primary
  * 04* MEMORY MODE: Non-XMEM
  * 04* DISPATCH MODE: Task
  * 04* RECOVERY TYPE: None
  * 04* ADDRESS SPACE: Caller's
* 03* EXECUTION
  * 04* MODE: Enabled
  * 04* STATE: Supervisor
  * 04* KEY: 0
  * 04* AMODE: 64
  * 04* LOCKS OBTAINED: None
  * 04* ASCMODE: Primary
  * 04* MEMORY MODE: Non-XMEM
  * 04* ADDRESS SPACE: Caller's
* 02* LINKAGE: Branched to.
* 03* CALLERS:
  * Any
* 02* INPUT:
* 03* ENTRY-REGISTERS:
  * 03* R0 - R12,R15 - Irrelevant
  * AR0-AR15 - Irrelevant
* 02* OUTPUT:
* 02* EXIT-NORMAL: RETURN TO CALLER
```

1206  z/OS V2R1.0 MVS Authorized Assembler Services Reference EDT-DXG
ISGLCR64 callable service

*03* CONDITIONS: successful completion

*03* EXIT-REGISTERS:
  * R0 - R14 - Unchanged
  * R15 - Return code (always 0)
  * AR0-AR15 - Irrelevant

*03* RETURN-CODES: None

*02* EXIT-ERROR: None

***********************************************************************

*01* TERMINATION-CONSIDERATIONS: None

***********************************************************************

*01* EXTERNAL-REFERENCES :

*02* ROUTINES: Latch Services.

*02* DATA-AREAS: None

*02* CONTROL-BLOCKS:
  * Name Use Mapping Description
  * ISGYLID CW ISGLMASM Latch Identity Entry
  * Legend: C=Create, R=Read, W=Write, D=Delete

*01* TABLES: TRTABLE

*01* MACROS-EXECUTABLE:
  * None

*01* SERIALIZATION:
  * None

*01* MESSAGES:
  * None

*01* POST-CODES:
  * None

*01* ABEND-CODES:
  * None

*01* WAIT-STATE-CODES:
  * None

*01* CHANGE-ACTIVITY:
  * None
ISGLCR64 callable service

CALL ISGLCR64,(NUM_LATCH,LS_NAME,PRIVATE,LS_TOKEN,RETCD),X MF=(E,CREATE_DPL)

* Initialize Storage for initial LID Entry Block.
* Note that in this example the DYNASTOR section is not freed and
  non-pertinent data is placed in the DYNASTOR section for simplicity.
* The default subpool associated with the STORAGE OBTAIN macro has a
  lifetime of the address space, so the DYNASTOR * section will exist
  for the life of the address space as well.
* However, IBM recommends using a separate storage request
  for the LIDPointerArray and ISGYLID_Entry blocks so that only
  necessary data will continue to exist beyond the life of the calling
  module.
* Also, it is not necessary to initialize all latch IDs in the set
  to point to a default ISGYLID_ENTRY block. NULL values in the
  Latch ID Pointer Array are acceptable.

* Set Latch ID array entries all to address of Initial Lid Entry ---
  indicating all entries are currently unused.

XC INIT_STOR,INIT_STOR
XC ENTIRELIDARRAY,ENTIRELIDARRAY
LA R3,INIT_STOR Base ISGYLID ENTRY block at address of allocated storage
USING ISGYLID_ENTRY, R3
LA R4,INITLIDSTR
STG R4,LIDPRINTABLESTRING@
LHI R4,L'INITLIDSTR' Put length of INITLIDSTR into entry block
STH R4,LIDPRINTABLESTRINGLENGTH
L R4,ONEMINUTE Set hold threshold value to one minute
ST R4,LIDHOLDTHRESHOLD
L R4,THIRTYSECONDS Set contention threshold value to thirty seconds
ST R4,LIDCONTHRESHOLD

* Set Latch ID array entries all to address of Initial Lid Entry ---
* Latch ID Pointer Array must be in the primary address space.

*******************************************************************************

LGH R6,0
L R4,NUM_LATCH

LOOP1 EQU *
STG R3,LIDPTRARRAY(R6)
A R6,LIDPTRARRAYLEN
BCT R4,LOOP1
DROP R3

*******************************************************************************

* Attach Latch IDs to Latch Set

*******************************************************************************

CALL ISGLID64,(LS_TOKEN,LIDPTRARRAY,LIDVERSION, X
RETCD),MF=(E,LID_DPL)

*******************************************************************************

* Change Latch ID Entry for latch #3

* Note:
* Once the Latch Identity Pointer Array has been attached to the
* latch set, it cannot be deleted. However it can be replaced by
* calling the service again and specifying a new array. To change
* the Latch Identity for a particular latch, allocate a new latch
* identity block, fill it out, and update the corresponding array
* entry. Then the program is free to delete the previous latch
* identity block.

*******************************************************************************

MVC NEW_STOR,INIT_STOR Copy LID Entry to new storage
USING ISGLID_ENTRY,R3 Use R3 as base for LID Entry Block
LA R3,NEW_STOR
LA R4,NEWLIDISTR
STG R4,LIDPRINTABLESTRING@ Store address and length of new X
string in ISGLID_ENTRY block
LHI R5,L'NEWLIDISTR
STH R5,LIDPRINTABLESTRINGLENGTH
L R4,LATCH_NUM Point to new ISGLID_ENTRY block
SLL R4,3 Multiply by 8 (size of LIDPTRARRAY)
STG R3,LIDPTRARRAY(R4)
DROP R3

*******************************************************************************

* Obtain latch #3

*******************************************************************************

MVC REQ_IDH,PSAAOLD-PSA Use ASCB address as high half
MVC REQ_IDL,PSATOLD-PSA Use TCB address as low half
CALL ISGLOB64,(LS_TOKEN,LATCH_NUM,REQ_ID,SUSPEND,EXCLUSIVE, X
ECB_ADDR,LATCH_TOKEN1,WORK_AREA,RETCD), X
MF=(E,OBTAIN_DPL)

*******************************************************************************

* Release latch #3

*******************************************************************************

CALL ISGLRE64,(LS_TOKEN,LATCH_TOKEN1, X
UNCOND,WORK_AREA,RETCD),MF=(E,RELEASE_DPL)

*******************************************************************************

Chapter 121. ISGLCR64 — Create a latch set in 64-bit mode 1209
ISGLCR64 callable service

* Purge requestor from latch set.
* Normally reserved for recovery situations.
* **********************************************************************
  * XC REQ_IDL,REQ_IDL Clear the low half of requestor ID
  * CALL ISGLPR64,(LS_TOKEN,REQ_ID,RETCD),MF=(E,PURGE_DPL)
  * **********************************************************************
  * Purge all granted and pending requests for a group of requestors for
  * a group of latch sets in the current address space.
  * Normally reserved for recovery situations.
  * **********************************************************************
  * CALL ISGLPB64,(=AD(0),REQ_ID,REQ_MASK,LS_NAME,LS_MASK,RETCD), X
  * MF=(E,PURGEBA_DPL)
  * **********************************************************************
* Exit
* Restore caller's regs and return. Also restores caller's PSW key and
* State without a MODESET MODE=PROB.
* **********************************************************************
  * LGHI R15,0 Set return code to zero
  * PR ,
  * **********************************************************************
* Equates
* **********************************************************************
  * R0 EQU 0
  * R1 EQU 1
  * R2 EQU 2
  * R3 EQU 3
  * R4 EQU 4
  * R5 EQU 5
  * R6 EQU 6
  * R7 EQU 7
  * R8 EQU 8
  * R9 EQU 9
  * R10 EQU 10
  * R11 EQU 11
  * R12 EQU 12
  * R13 EQU 13
  * R14 EQU 14
  * R15 EQU 15
  *
  * **********************************************************************
* Constants
* **********************************************************************
  * NUM_LATCH DC F'16' Number of latches to create - input
to create
  * LS_NAME DC CL48'EXAMPLE.ONE_LATCH_SET' latch set name - input to
create
  * LS_MASK DC 48XL1'FF' Latch set name mask to match all
  * REQ_MASK DC 4XL1'FF' Requestor ID mask to match all
  DC 4XL1'00'
**ISGLCR64 callable service**

* requestors with the same first
  half (ascb@) and any second half
INITLIDSTR DC C'Latch not used' Initial LID string and length
* LIDVERSION DC AL1(ISGYLID_VERSION1) Set latch version to default
* LIDPTRARRAYLEN DC F'8'
* THIRTYSECONDS DC F'30'
* ONEMINUTE DC F'60'
NEWLIDSTR DC C'Serializing Abstract Resource' New string and length
PRIVATE DC A(ISGCR64_PRIVATE) Create option - input to create -
  (defined in IDF)
LATCH_NUM DC F'3' Number of latch to be obtained -
  input to obtain
SUSPEND DC A(ISGLAT_SYNC) Obtain option - input to obtain -
  (defined in IDF)
EXCLUSIVE DC A(ISGLAT_EXCLUSIVE) access option - input to obtain -
  (defined in IDF)
UNCOND DC A(ISGLAT_COND) Release option - input to release -
  (defined in IDF)
**********************************************************************
* Dynamic area for save area, parmlists, and variables
**********************************************************************

* DYNASTORE DSECT
* SAVEAREA DS 18D 144-byte register save area
* ENTIRELIDARRAY DS OXL128
* LIDPTRARRAY DS 16AD LIDPtrArray
* CREATE_DPL DS OF
  CALL ,(NUM_LATCH,LS_NAME,PRIVATE,LS_TOKEN,RETCD),MF=L
* OBTAIN_DPL DS OF
  CALL ,(LS_TOKEN,LATCH_NUM,REQ_ID,SUSPEND,EXCLUSIVE,
        ECB_ADDR,LATCH_TOKEN1,WORK_AREA,RETCD),MF=L
* LID_DPL DS OF
  CALL ,(LS_TOKEN,LIDPTRARRAY,LIDVERSION,
        RETCD),MF=L
* RELEASE_DPL DS OF
  CALL ,(LS_TOKEN,LATCH_TOKEN1,UNCOND,WORK_AREA,RETCD),MF=L
* PURGE_DPL DS OF
  CALL ,(LS_TOKEN,REQ_ID,RETCD),MF=L
* PURGEBA_DPL DS OF
  CALL ,(0,REQ_ID,REQ_MASK,LS_NAME,LS_MASK,RETCD),MF=L
* ECB DS F ECB (used only when the obtain
  option is ISGLAT_ASYNC_ECB)
ECB_ADDR DS A Address of ECB - input to obtain -
  (required for the interface, but
  only used when obtain option is
  ISGLAT_ASYNC_ECB)
LS_TOKEN DS 2F Latch set token - output from create
  and input to obtain, latch identity,
  release, purge, and purge group
* LATCH_TOKEN1 DS 2F Latch token - output from obtain
  and input to release
### ISGLCR64 callable service

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ_ID</td>
<td>DS 8D</td>
<td>Double word requestor ID - input to obtain, purge, and purge group</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_IDH</td>
<td>DS F</td>
<td>First half of requestor ID (ascb@)</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REQ_IDL</td>
<td>DS F</td>
<td>Second half of requestor ID (tcb@)</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORK_AREA</td>
<td>DS 64D</td>
<td>512-byte work area</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETCD</td>
<td>DS F</td>
<td>Return code - output from services</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INIT_STOR</td>
<td>DS XL(ISGLID_ENTRY_LEN)</td>
<td>Isgylid_entry block storage</td>
</tr>
</tbody>
</table>

#### Notes
- IHAPSA Needed for ascb@ and tcb@ req ID
- ISGLMASM Needed for latch constants
- END LATCHX64
Chapter 122. ISGLID64 — Identify a latch set in 64-bit mode

Description
The ISGLID64 callable service provides a latch set creator the ability to attach a latch identity array (in ISGLMASM or ISGLMC) to the latch set for the purposes of identifying the individual latches in the latch set. The LIDArray must be in the primary space. The following callable services are related to the ISGLID64 service:
- ISGLCRT
- ISGLCRT64
- ISGLID
- ISGLID64

Environment
The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PKM allowing key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Primary= the space of the latch set creator</td>
</tr>
<tr>
<td>AMODE</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be accessible from the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements
After the latch identity pointer array has been attached to the latch set, the attached LIDArray cannot be deleted. However, the LIDArray can be replaced by calling the service again and specifying a new array. To change the latch identity for a particular latch, allocate a new latch identity block, fill it out, and update the corresponding array entry. Then, you can delete the previous latch identity block.

Restrictions
None.

Input register information
Before calling the ISGLID64 service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 144-byte save area located in the primary address space</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:
ISGLID64 callable service

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
</tr>
<tr>
<td>2-13</td>
</tr>
<tr>
<td>14-15</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

Register

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
</tr>
<tr>
<td>2-13</td>
</tr>
<tr>
<td>14-15</td>
</tr>
</tbody>
</table>

Performance implications

None

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- `lsetToken`
- `LIDPtrArray`
- `LIDEntryVersion`

ISGLID64 returns values in the following parameter:

- `retcode`

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL ISGLID64</td>
<td><code>,(lsetToken,LIDPtrArray,LIDEntryVersion,retcode)</code></td>
</tr>
</tbody>
</table>

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

- **lsetToken**
  
  Specifies an 8-character field that contains the latch set token returned from ISGLCRT.

- **LIDPtrArray**
  
  Specifies the latch identity pointer array.

- **LIDEntryVersion**
  
  Specifies a 1-byte area that contains the version of the LID entries.
ISGLID64 callable service

**RetCode**

Specifies a 4-byte or 32-bit area that contains return code from the ISGLID64 service.

**ABEND codes**

The caller might encounter abend code X'9C6' for certain errors. See **z/OS MVS System Codes** for explanations and responses for these codes.

**Return codes**

When the ISGLID64 service returns control to your program, the RetCode parameter contains a hexadecimal return code. The following table identifies the hexadecimal reason codes and meaning associated with each reason code:

<table>
<thead>
<tr>
<th>Table 108. ISGLID64 Return Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Return code and Equate symbol</strong></td>
</tr>
<tr>
<td>00000000</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>xxxx0401</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Example**

See “LATCHX64 - How to call AMODE 64 latch services” on page 1205.
Chapter 123. ISGLOB64 — Obtain a latch in 64-bit mode

Description

Call the 64-bit Latch_Obtain service to request exclusive or shared ownership of a latch. When a requestor owns a particular latch, the requestor can use the resource associated with that latch. The following callable services are related to Latch_Obtain:

ISGLCR64
Creates a latch set that an application can use to serialize resources.

ISGLRE64
Releases ownership of an owned latch or a pending request to obtain a latch.

ISGLPR64
Purges all granted and pending requests for a particular requestor within a specific latch set.

In the following description of 64-bit Latch_Obtain:

• The term requestor describes a task or SRB routine that calls the Latch_Obtain service to request ownership of a latch.

• Equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Obtain. For example, “ISGLOBT_COND (value of 1)” indicates the equate symbol ISGLOBT_COND and its associated value, 1.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PKM allowing key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be accessible from the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

If you specify an obtain_option of ISGLOBT_ASYNC_ECB (as described in "Parameters" on page 1219), initialize the ECB pointed to by the value on the ECB_address parameter to zero before calling Latch_Obtain.

Before you use the Latch_Obtain service, you need to include the ISGLMASM macro to obtain assembler declaration statements for Latch_Obtain. ISGLMASM provides the following equate symbols for use when calling Latch_Obtain:

* Latch Obtain Option Equate Symbols
*
ISGLOB64 callable service

ISGLOBT_SYNC  EQU  0
ISGLOBT_COND   EQU  1
ISGLOBT_ASYNC_ECB  EQU  2

*  *  Latch Obtain Access Equate Symbols  *
*  *
ISGLOBT_EXCLUSIVE  EQU  0
ISGLOBT_SHARED    EQU  1
*
*  *  Latch Obtain Equate Symbols  *
*  *
ISGLOBT_SUCCESS   EQU  0
ISGLOBT_CONTENTION EQU  4
*

Restrictions
1. The caller of the 64-bit Latch_Obtain must have a PSW key that allows access
to the latch set storage.
2. The ECB specified on the ECB_address parameter must reside in storage with a
storage key that matches the latch set storage key.
3. You must call the 64-bit Latch_Obtain from the same primary address space
from which the 64-bit Latch_Create service was called.

Input register information
Before calling the 64-bit Latch_Obtain service, the caller must ensure that the
following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 144-byte save area located in the primary address space</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications
See the information about obtaining latches in z/OS MVS Programming: Authorized
Assembler Services Guide for performance implications related to the 64-bit
Latch_Obtain service.
ISGLOB64 callable service

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:
- `latch_set_token`
- `latch_number`
- `requestor_ID`
- `obtain_option`
- `access_option`
- `ECB_address`

The 64-bit Latch_Obtain returns values in the following parameters:
- `latch_token`
- `return_code`

The 64-bit Latch_Obtain uses the following parameter for temporary storage:
- `work_area`

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL ISGLOB64</td>
<td>/latch_set_token, /latch_number, /requestor_ID, /obtain_option, /access_option, /ECB_address, /latch_token, /work_area, /return_code</td>
</tr>
</tbody>
</table>

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

- **latch_set_token**
  - Specifies an 8-byte area that contains the latch_set_token that the 64-bit Latch_Create service returned earlier when it created the latch set.

- **latch_number**
  - Specifies a fullword integer that contains the number of the latch to be obtained. The latch_number must be in the range from 0 to the total number of latches in the associated latch set minus one.

- **requestor_ID**
  - Specifies an 8-byte area that contains a value that identifies the caller of the 64-bit Latch_Obtain service. The requestor_ID can be any value except all binary zeros.

Recovery routines can purge all granted and pending requests for a particular requestor (identified by a requestor_id) within a specific latch set. When specifying the requestor_ID on the 64-bit Latch_Obtain, consider which latches would be purged if the 64-bit Latch_Purge service were to be called with the
ISGLOB64 callable service

specified requestor_ID. For more information about the 64-bit Latch_Purge service, see Chapter 125, “ISGLPR64 — Purge a requestor from a latch set in 64-bit mode,” on page 1229.

,obtain_option
A fullword integer that specifies how the system is to handle the 64-bit Latch_Obtain request if the latch manager cannot immediately grant ownership of the latch to the requestor:

ISGLOB64_SYNC (value of 0)
The system processes the request synchronously. The system suspends the requestor. When the latch manager eventually grants ownership of the latch to the requestor, the system returns control to the requestor.

ISGLOB64_COND (value of 1)
The system processes the request conditionally. The system returns control to the requestor with a return code of ISGLOBT_CONTENTION (value of 4). The latch manager does not queue the request to obtain the latch.

ISGLOB64_ASYNC_ECB (value of 2)
The system processes the request asynchronously. The system returns control to the requestor with a return code of ISGLOBT_CONTENTION (value of 4). When the latch manager eventually grants ownership of the latch to the requestor, the system posts the ECB pointed to by the value specified on the ECB_address parameter.

When you specify this option, the ECB_address parameter must contain the address of an initialized ECB that is addressable from the home address space (HASN).

,access_option
A fullword or character string that specifies the access required:
• ISGLOBT_EXCLUSIVE (value of 0) - Exclusive (write) access
• ISGLOBT_SHARED (value of 1) - Shared (read) access

,ECB_address
Specifies a fullword that contains the address of an ECB. If you specify an obtain_option of ISGLOB64_SYNC (value of 0) or ISGLOB64_COND (value of 1) on the call to Latch_Obtain, the ECB_address field must be valid (though its contents are ignored). IBM recommends that an address of 0 be used when no ECB is to be processed.

If you specify an obtain_option of ISGLOB64_ASYNC_ECB (value of 2) and the system returns a return code of ISGLOBT_CONTENTION (value of 4) to the caller, the system posts the ECB pointed to by the value specified on the ECB_address parameter when the latch manager grants ownership of the latch to the requestor.

,latch_token
Specifies an 8-byte area to contain the latch token returned by the 64-bit Latch_Obtain service. You must provide this value as a parameter on a call to the 64-bit Latch_Release service to release the latch.

,work_area
Specifies a 512-byte work area that provides temporary storage for the 64-bit Latch_Obtain service. The work area should begin on a doubleword boundary to optimize performance. The work area must be in the same storage key as the caller of 64-bit Latch_Obtain.
ISGLOB64 callable service

,.return_code
Specifies a fullword integer that is to contain the return code from the 64-bit Latch_Obtain service.

ABEND codes
The caller might encounter abend code X’9C6’ for certain errors. See z/OS MVS System Codes for explanations and responses for these codes.

Return codes
When the 64-bit Latch_Obtain service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 109. ISGLOBT64 Return Codes

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (0) ISGLOBT_SUCCESS</td>
<td>Meaning: The Latch_Obtain service completed successfully. Action: None.</td>
</tr>
<tr>
<td>04 (4) ISGLOBT_CONTENTION</td>
<td>Meaning: A requestor called Latch_Obtain with an obtain_option of ISGLOBT_COND (value of 1) or ISGLOBT_ASYNC_ECB (value of 2). The latch is not immediately available. Action: If the requestor specified an obtain_option of ISGLOBT_COND (value of 1), no response is required. If the requestor specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2), and the latch is still required, wait on the ECB to be posted when the latch manager grants ownership of the latch to the requestor.</td>
</tr>
</tbody>
</table>

Example
See “LATCHX64 - How to call AMODE 64 latch services” on page 1205 for an example of how to call Latch_Obtain in assembler language.
Chapter 124. ISGLRE64 — Release a latch in 64-bit mode

Description

Call the 64-bit Latch_Release service to release ownership of an owned latch or a pending request to obtain a latch. Requestors should call 64-bit Latch_Release when the use of a resource associated with a latch is no longer required. The following callable services are related to 64-bit Latch_Release:

**ISGLCR64**
- Creates a 64-bit latch set that an application can use to serialize resources.

**ISGLOB64**
- Requests exclusive or shared control of a 64-bit latch.

**ISGLPR64**
- Purges all granted and pending requests for a particular requestor within a specific 64-bit latch set.

In the following description of the 64-bit Latch_Release:

- The term *requestor* describes a program that calls the Latch_Release service to release ownership of an owned latch or a pending request to obtain a latch.
- Equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to the 64-bit Latch_Obtain. For example, “ISGLREL_COND (value of 1)” indicates the equate symbol ISGLREL_COND and its associated value, 1.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PSW allowing key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Control parameters must be accessible from the primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

Before you use the 64-bit Latch_Release service, include the ISGLMASM macro to obtain assembler declaration statements for Latch_Release. ISGLMASM provides the following equate symbols for use when calling the 64-bit Latch_Release:

* Latch Release Option Equate Symbols
  * ISGLREL_UNCOND    EQU 0
  * ISGLREL_COND      EQU 1

* Latch Release Return Codes
  * ISGLREL_SUCCESS   EQU 0
ISGLRE64 callable service

Restrictions
1. The caller of the 64-bit Latch_Release must have a PSW key that allows access to the latch set storage.
2. You must call the 64-bit Latch_Release from the same primary address space from which the 64-bit Latch_Create service was called.

Input register information
Before calling the 64-bit Latch_Release service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 144-byte save area located in the primary address space</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications
See the information about releasing latches in z/OS MVS Programming: Authorized Assembler Services Guide for performance implications related to the 64-bit Latch_Release service.

Syntax
Write the CALL as shown on the syntax diagram, coding all parameters in the specified order.

Assign values to the following parameters:
- latch_set_token
- latch_token
- release_option
ISGLRE64 callable service

The 64-bit Latch_Release returns a value in the following parameter:
  * return_code

The 64-bit Latch_Release uses the following parameter for temporary storage:
  * work_area

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL ISGLRE64</td>
<td>(latch_set_token, latch_token, release_option, work_area, return_code)</td>
</tr>
</tbody>
</table>

**Parameters**

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

**latch_set_token**

Specifies an 8-byte area that contains the latch set token returned to the caller of the 64-bit Latch_Create service. The latch set token identifies the latch set that contains the latch to be released.

**latch_token**

Specifies an 8-byte area that contains the latch token returned to the caller of the 64-bit Latch_Obtain service. The latch token identifies the request to be released.

**release_option**

Specifies a fullword integer that tells the latch manager what to do when the requestor either no longer owns the latch to be released or still has a pending request to obtain the latch to be released:

**ISGLREL_UNCOND (value of 0)**

Abend the requestor:

- If a requestor originally specified an obtain_option of ISGLOBT_SYNC (value of 0) when obtaining the latch, the latch manager does not release the latch. The system abends the caller of Latch_Release with abend X'9C6', reason code xxxx0009.
- If a requestor originally specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) when obtaining the latch, the latch manager does not release the latch. The system abends the caller of Latch_Release with abend X'9C6', reason code xxxx0007.
- If the latch manager does not find a previous Latch_Obtain request for the specified latch, the system abends the caller of Latch_Release with abend X'9C6', reason code xxxx000A.

**ISGLREL_COND (value of 1)**

Return control to the requestor:

- If a requestor originally specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) when obtaining the latch, the latch manager releases the request for ownership of the latch. The system returns control to the caller of Latch_Release with a return code of ISGLREL_NOT_OWNED_ECB_REQUEST (value of 4).
ISGLRE64 callable service

- If a requestor originally specified an obtain_option of ISGLOBT_SYNC (value of 0) when obtaining the latch, the latch manager does not release the request for ownership of the latch. The system returns control to the caller of Latch_Release with a return code of ISGLREL_STILL_SUSPENDED (value of 8).
- If the latch manager does not find a previous Latch_Obtain request for the specified latch, the system returns control to the caller of Latch_Release with a return code of ISGLREL_INCORRECT_LATCH_TOKEN (value of 12).

\[ \text{work_area} \]
Specifies a 512-byte work area that provides temporary storage for the 64-bit Latch_Release service. The work area should begin on a doubleword boundary to optimize performance. The work area must be in the same storage key as the caller of Latch_Release.

\[ \text{return_code} \]
Specifies a fullword integer that is to contain the return code from the 64-bit Latch_Release service.

**ABEND codes**

The caller might encounter abend code X'9C6' for certain errors. See [z/OS MVS System Codes](https://www.ibm.com/mvs/library) for explanations and responses.

**Return codes**

When the 64-bit Latch_Release service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00 (0) ISGLREL_SUCCESS        | **Meaning:** The Latch_Release service completed successfully. The caller released ownership of the specified latch request.  
**Action:** None. |
| 04 (4) ISGLREL_NOT_OWNED_ECB_REQUEST | **Meaning:** The requestor that originally called the Latch_Obtain service is still expecting the system to post an ECB (to indicate that the requestor has obtained the latch). The call to the Latch_Release service specified a release_option of ISGLREL_COND (value of 1). The latch manager does not post the ECB at the address specified on the original call to Latch_Obtain. The latch manager releases the latch.  
**Action:** Validate the integrity of the resource associated with the latch (the requestor might have used the resource without waiting on the ECB). If the resource is undamaged, no action is necessary (a requestor routine may have been in the process of cancelling the request to obtain the latch). |
### ISGLRE64 callable service

#### Table 110. ISGLRE64 Return Codes (continued)

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08 (8) ISGLREL_STILL_SUSPENDED | **Meaning:** Program error. The request specified a correct latch token, but the program that originally requested the latch is still suspended and waiting to obtain the latch.

The latch requestor originally specified an obtain_option of ISGLOBT_SYNC on the call to the Latch_Obtain service. The call to the Latch_Release service specified a release_option of ISGLREL_COND (value of 1). The latch manager does not release the latch. The latch requestor remains suspended.

**Action:**
- Wait for the latch requestor to obtain the latch and receive control back from the system, then call the Latch_Release service again, or
- End the program that originally requested the latch.

| 0C (12) ISGLREL_INCORRECT_LATCH_TOKEN | **Meaning:** The latch manager could not find a granted or pending request associated with the value on the latch token parameter. The latch manager does not release a latch.

This return code does not indicate an error if a routine calls Latch_Release to ensure that a latch is released. For example, if an error occurs when a requestor calls the Latch_Obtain service, the requestor’s recovery routine might call Latch_Release to ensure that the requested latch is released. If the error prevented the requestor from obtaining the latch, the recovery routine receives this return code.

**Action:** If the return code is not expected, validate that the latch token is the same latch token returned to the caller of Latch_Obtain.

#### Example

See “LATCHX64 - How to call AMODE 64 latch services” on page 1205 for an example of how to call the 64-bit Latch_Release in assembler language.
Chapter 125. ISGLPR64 — Purge a requestor from a latch set in 64-bit mode

Description

Call the 64-bit Latch_Purge service to purge all granted and pending requests for a particular requestor within a specific 64-bit latch set. Recovery routines should call 64-bit Latch_Purge when one or more errors prevent requestors from releasing latches. The following callable services are related to the 64-bit Latch_Purge:

ISGLCR64
Creates a 64-bit latch set that an application can use to serialize resources.

ISGLOB64
Requests exclusive or shared control of a 64-bit latch.

ISGLRE64
Releases control of an owned 64-bit latch or a pending request to obtain a 64-bit latch.

In the following description of 64-bit Latch_Purge, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to 64-bit Latch_Purge.

Environment

The requirements for the caller are:

Environmental factor | Requirement
--- | ---
Minimum authorization | Supervisor state or PSW allowing key 0-7
Dispatchable unit mode | Task or SRB
Cross memory mode | Any PASN, any HASN, any SASN
AMODE | 64-bit
ASC mode | Primary
Interrupt status | Enabled for I/O and external interrupts
Locks | No locks held
Control parameters | Control parameters must be accessible from the primary address space.

Programming requirements

Before you use the 64-bit Latch_Purge service, you need to include the ISGLMASM macro to obtain assembler declaration statements for Latch_Purge. ISGLMASM provides the following equate symbols for use when calling Latch_Purge:

* Latch Purge Return Codes
* ISGLPRG_SUCCESS EQU 0
ISGLPRG_DAMAGE_DETECTED EQU 4
*

Restrictions

1. The caller of 64-bit Latch_Purge must have a PSW key that allows access to the latch set storage.
ISGLPR64 callable service

2. You must call the 64-bit Latch_Purge from the same primary address space from which the 64-bit Latch_Create service was called.

Input register information
Before calling the 64-bit Latch_Purge service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 144-byte save area located in the primary address space</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications
None.

Syntax
Write the CALL as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:
- `latch_set_token`
- `requestor_ID`

The 64-bit Latch_Purge returns a value in the `return_code` parameter.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>
| CALL ISGLPR64   | `latch_set_token`
|                 | `requestor_ID`
|                 | `return_code` |
Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch_set_token previously returned by the Latch_Create service. The latch set token identifies the latch set from which latch requests are to be purged.

requestor_ID
Specifies an 8-byte area that contains the requestor_ID originally specified on one or more previous calls to the Latch_Obtain service. The Latch_Purge service is to release all Latch_Obtain requests that specify this requestor_ID.

return_code
A fullword integer that contains the return code from the Latch_Purge service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See z/OS MVS System Codes for explanations and responses.

Return codes

When the 64-bit Latch_Purge service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 111. ISGLPRG Return Codes

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(0)</td>
<td>ISGLPRG_SUCCESS</td>
<td>Meaning: The Latch_Purge service completed successfully.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>04(4)</td>
<td>ISGLPRG_DAMAGE_DETECTED</td>
<td>Meaning: Program error. While purging all requests for a particular requestor from a latch set, the latch manager found incorrect data in one or more latches. The latch manager tries to purge the latches that contain incorrect data, but the damage might prevent the latch manager from purging those latches. The latch manager purges the remaining latches (those with correct data) for the specified requestor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Take a dump and check for a storage overlay. If your application can continue without the resources serialized by the damaged latches, no action is required.</td>
</tr>
</tbody>
</table>

Example

See “LATCHX64 - How to call AMODE 64 latch services” on page 1205 for an example of how to call 64-bit Latch_Purge in assembler language.
Call the 64-bit Latch_Purge_by_Address_Space service to purge all granted and pending requests for a group of requestors for a group of latch sets in the same address space. To effectively use this service, your latch_set_names and your requestor_IDs should be defined such that they have a common portion and a unique portion. Groups of latch sets can then be formed by masking off the unique portion of the latch_set_name, and groups of latch requests in a latch set can then be formed by masking off the unique portion of the requestor_ID. Masking off the unique portion of the requestor_ID allows a single purge request to handle multiple latch sets and multiple requests in a latch set. Recovery routines should call 64-bit Latch_Purge_by_Address_Space when one or more errors prevent requestors from releasing latches.

The following callable services are related to 64-bit Latch_Purge_by_Address_Space:

**ISGLCR64**
- Creates a 64-bit latch set that an application can use to serialize resources.

**ISGLOB64**
- Requests exclusive or shared control of a 64-bit latch.

**ISGLRE64**
- Releases control of an owned 64-bit latch or a pending request to obtain a 64-bit latch.

**ISGLPR64**
- Purges all granted and pending requests for a particular requestor within a specific 64-bit latch set.

In the following description of the 64-bit Latch_Purge_by_Address_Space, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to the 64-bit Latch_Purge_by_Address_Space.

**Environment**

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Supervisor state or PSW allowing key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be accessible from the primary address space.</td>
</tr>
</tbody>
</table>
Programming requirements

Before you use the 64-bit Latch_Purge_by_Address_Space service, you need to include the ISGLMASM macro to obtain assembler declaration statements for the 64-bit Latch_Purge_by_Address_Space. ISGLMASM provides the following equate symbols for use when calling the 64-bit Latch_Purge_by_Address_Space:

* Latch Purge by Address Space Return Codes
* ISGLPRG_SUCCESS EQU 0
ISGLPRG_DAMAGE_DETECTED EQU 4
ISGLPRG_INCORRECT_MASK EQU C
* 

Restrictions

1. The caller of the 64-bit Latch_Purge_by_Address_Space must have a PSW key that allows access to the latch set storage.
2. You must call the 64-bit Latch_Purge_by_Address_Space from the same primary address space from which the 64-bit Latch_Create service was called.

Input register information

Before calling the 64-bit Latch_Purge_by_Address_Space service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 144-byte save area located in the primary address space</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications

None.
ISGLPB64 callable service

Syntax

Write the CALL as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- `latch_set_token`
- `requestor_ID`
- `requestor_ID_mask`
- `latch_set_name`
- `latch_set_name_mask`

The 64-bit Latch_Purge_by_Address_Space returns a value in the return_code parameter.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL ISGLPB64</td>
<td><code>(latch_set_token, requestor_ID, requestor_ID_mask, latch_set_name, latch_set_name_mask, return_code)</code></td>
</tr>
</tbody>
</table>

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

**latch_set_token**

Specifies an 8-byte area that contains the latch_set_token previously returned by the Latch_Create service or a value of zero. If the value is not zero, the latch_set_token identifies the latch set from which latch requests are to be purged. If the latch_set_token is set to zero, a group of latch sets, determined by the latch_set_name and latch_set_name_mask, will have their latch requests purged.

**requestor_id**

Specifies an 8-byte area that contains a portion of the requestor_ID originally specified on one or more previous calls to the Latch_Obtain service. This operand will be compared to the result of logically ANDing each requestor_ID in the latch set with the requestor_ID_mask. Make sure that any corresponding bits that are zero in the requestor_ID_mask are also zero in this field, otherwise no ID matches will occur. Each requestor_ID that has a name match will have its Latch_Obtain requests released.

**requestor_ID_mask**

Specifies an 8-byte area that contains the requestor_ID_mask that will be logically ANDed to each requestor_ID in the latch set and then compared to the requestor_ID operand. Each requestor_ID that has a name match will have its Latch_Obtain requests released.

**latch_set_name**

Specifies a 48-byte area that contains the portion of the latch_set_name that will be compared to the result of logically ANDing the latch_set_name_mask
with each latch set name in the primary address space. Make sure that any corresponding bits that are zero in the latch_set_name_mask are also zero in this field, otherwise no name matches will occur. Each latch set that has a name match will have its Latch_Obtain requests released. If the latch_set_token operand is non-zero this operand is ignored.

**latch_set_name_mask**
Specifies a 48-byte area that contains the mask that will be logically ANDed to each of the latch set names in the primary address space and then compared to the latch_set_name operand. Each latch set that has a name match will have its Latch_Obtain requests released. If the latch_set_token operand is non-zero this operand is ignored.

**return_code**
A fullword integer that contains the return code from the Latch_Purge_By_Address_Space service.

### ABEND codes
The caller might encounter abend code X'9C6' for certain errors. See [z/OS MVS System Codes](z/OS MVS System Codes) for explanations and responses.

### Return codes
When the 64-bit Latch_Purge_by_Address_Space service returns control to your program, the return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

<table>
<thead>
<tr>
<th>Return code and Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0 (0) ISGLPRG_SUCCESS         | **Meaning:** The Latch_Purge_by_Address_Space service completed successfully.  
**Action:** None. |
| 04 (4) ISGLPRG_DAMAGE_DETECTED| **Meaning:** Program error. While purging all requests for a particular requestor from a latch set, the latch manager found incorrect data in one or more latches. The latch manager tries to purge the latches that contain incorrect data, but the damage might prevent the latch manager from purging those latches. The latch manager purges the remaining latches (those with correct data) for the specified requestor.  
**Action:** Take a dump and check for a storage overlay. If your application can continue without the resources serialized by the damaged latches, no action is required. |
Chapter 127. ISGQUERY — Global resource serialization query service

Description

The GRS query service routine is given control from the ISGQUERY macro to:
• Search a resource name list (RNL) for a given QNAME/RNAME pair.
• Obtain information on resources and requesters of outstanding ENQ requests.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state. Any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31- or 64-bit</td>
</tr>
<tr>
<td></td>
<td>If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td></td>
<td>If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>For REQINFO=RNLSEARCH, the caller may be unlocked or hold both a local lock (LOCAL or CML) and the CMSEQDQ lock.</td>
</tr>
<tr>
<td></td>
<td>For all other REQINFO requests, the caller must not hold any locks.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller’s dispatchable unit access list (DU-AL).</td>
</tr>
<tr>
<td></td>
<td>The control parameters must be in the same key as the caller.</td>
</tr>
<tr>
<td></td>
<td>The user-provided answer area via the ANSAREA parameter has the same requirements and restrictions as the control parameters.</td>
</tr>
</tbody>
</table>

Programming requirements

The caller must include the ISGYQUAC macro to get a mapping for the answer area.

Note: The ISGYQUAA macro is stabilized as of z/OS R12.

The caller must include the ISGYCON macro to get the values for the return and reason codes.

The caller must include the ISGRNLE macro to get a mapping for the RNLE.
ISGQUERY macro

Restrictions

Do not issue ISGQUERY before the GRS address space has been initialized.

There is a restriction on the number of concurrent resource requests in an address space. These include unauthorized ISGENQ, ENQ, RESERVE, and incomplete GQSCAN and ISGQUERY requests. Reason code ISGQUERYRsn_MaxConcurrentRequests is issued if ISGQUERY would cause this limit to be exceeded.

When multilevel security support is active on the system, unauthorized callers of ISGQUERY who specify REQINFO=QSCAN must have at least READ authorization to the ISG.QSCANSERVICES.AUTHORIZATION resource in the FACILITY class. When multilevel security support is active on the system, unauthorized callers of ISGQUERY who specify REQINFO=LATCHCA must have at least READ authorization to the ISG.LATCHECASERVICES.AUTHORIZATION resource in the FACILITY class. You can activate the multilevel security support through the SETROPTS MLACTIVE option in RACF. For general information about defining profiles in the FACILITY class, see z/OS Security Server RACF Command Language Reference and z/OS Security Server RACF Security Administrator’s Guide. For information about multilevel security, see z/OS Planning for Multilevel Security and the Common Criteria.

Callers who specify REQINFO=LATCHCA must not hold any FRRs.

This macro supports multiple versions. Some keywords are unique to certain versions. For more information, see the description of the "PLISTVER=IMPLIED_VERSION" on page 1247 parameter and the common criteria.

Input register information

Before issuing the ISGQUERY macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR15 is not 0</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

In general, the narrower the search parameters (particularly QNAME and RNAME), the less time the query takes. Using both a specific QNAME and a specific RNAME gives better performance than using patterns.

The use of GATHERFROM=SYSPLEX might greatly degrade the performance of the query request.

Polling for ENQ contention through GQSCAN or ISGQUERY is not recommended. See the /OS MVS Planning: Global Resource Serialization and /OS MVS Programming: Authorized Assembler Services Guide for more information about monitoring contention through ENF 51.
ISGQUERY macro

Syntax

main diagram

```
name b ISGQUERY b

(REQINFO — RNLSEARCH) parameters-1
(REQINFO — ENQSTATS, ASID — osid, ANSAREA — ansarea)
(REQINFO — QSCAN) parameters-2
(REQINFO — LATCHCHECK) parameters-5
(REQINFO — USAGESTATS, ANSAREA — ansarea, ANSLEN — anslen)

(REQINFO — RNL) SIRNL, SERNL, RCRNL

(REQINFO — QNAME — qname, RNAME — rname, RNAMELEN — rnamelen)

(REQINFO — RNLE — rnle)

(parameters-1)

(parameters-2)

(parameters-3)

(SCANACTION — START)

(SCANACTION — RESUME, RESUMETOKEN — resumetoken, ANSAREA — ansarea, ANSLEN — anslen)

(SCANACTION — QUIT, RESUMETOKEN — resumetoken)

(RETCODE — reetcode, RSCODE — rsncode)

(MF — S)

(MF — (L, list addr, attr, COMPLETE))

(MF — (E, list addr, ))

(PLISTVER — IMPLIED_VERSION)

(PLISTVER — MAX)

(PLISTVER — 1)

(PLISTVER — 2)
```

parameters-1

parameters-2

parameters-3
Chapter 127. ISQUERY — Global resource serialization query service

parameters-3

- RESUMETOKEN resumetoken
- ANSAREA ansarea, ANSLEN anslen
- ANSDETAIL SUMMARY
- GATHERFROM SYSTEM
- ANSDETAIL FULL, GATHERFROM SYSPLEX
- ANSDETAIL FULL2
- ANSDETAIL FULL3
- REQUESTERLIMIT 32767
- SEARCH BY_ENQTOKEN, ENQTOKEN enqtoken

parameters-4

- QNAMEMATCH SPECIFIC, QNAME qname
- QNAMEMATCH PATTERN, QNAME qname
- RNAMEMATCH ANY
- RNAMEMATCH SPECIFIC, RNAME rname, RNAMENLEN rnamelen
- RNAMEMATCH PATTERN, RNAME rname, RNAMENLEN rnamelen
- SCOPE ANY
- SERIALIZEBY ANY
- SERIALIZEBY RESERVE
- SERIALIZEBY ENQ_ONLY
- SYSNAME ANY_SYSNAME
- SYSNAME sysname
- MINREQUESTERS NO_MINREQ
- MINREQUESTERS minrequesters
- ASID ANY_ASID
- ASID asid
- JOBNAME ANY_JOBNAME
- JOBNAME jobname
- TTOKEN ANY_TTOKEN
- TTOKEN ttoken
- MINOWNERS NO_MINOWN
- MINOWNERS minowners
- MINWAITERS NO_MINWAIT
- MINWAITERS minwaiters
- USERDATAMATCH ANY
- USERDATAMATCH SPECIFIC, USERDATA userdata
- USERDATAMATCH PATTERN, USERDATA userdata, USERDALEN userdatalen

parameters-5

- ANALYZE WAITER
- ANSAREA ansarea
- ANSLEN anslen
ISGQUERY macro

Parameters

The parameters are explained as follows:

name

An optional symbol, starting in column 1, that is the name on the ISGQUERY macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

\*ANALYZE=WAITER

When REQINFO=LATCHECAP is specified, a required output parameter, which queries LATCHECAP waiter data to determine if any long term latch contention exists that might be cause for concern. ISGQUERY only returns LATCHECAP data for waiters.

\*ANSAREA=ansarea

When REQINFO=ENQSTATS is specified, a required output parameter, which is to contain the returned information. The area is mapped by macro ISGYQUAA. A header area, mapped by DSECT ISGYQUAAHdr, is returned followed by additional data, two entries mapped by ISGYQUAASys and two entries mapped by ISGYQUAASp.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

\*ANSAREA=ansarea

When REQINFO=LATCHAREA is specified, a required output parameter, which is to contain the returned information. The area is mapped by macro ISGYQUAA. A header area, mapped by DSECT ISGYQUAAHdr, is returned followed by additional data mapped by ISGYQUAARs, ISGYQUAARsx, ISGYQUAARq, and ISGYQUAARqx. Note that the ANSDETAIL specified determines which data is returned.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

\*ANSAREA=ansarea

When REQINFO=LATCHECAP is specified, a required output parameter, which is to contain the returned information. The area is mapped by macro ISGYQUAA. A header area, mapped by DSECT ISGYQUAAHdr, is returned followed by additional data mapped by ISGYQUAAALd and ISGYQUAAALrd.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

\*ANSAREA=ansarea

When REQINFO=USAGESTATS is specified, a required output parameter, which is to contain the returned information. The area is mapped by macro ISGYQUAA. A header area, mapped by DSECT ISGYQUAAHdrUs, is returned followed by additional data mapped by ISGYQUAAUs.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

\*ANSAREA=ansarea

When SCANACTION=START and REQINFO=QSCAN are specified, a required output parameter, which is to contain the returned information. The area is mapped by macro ISGYQUAA. A header area, mapped by DSECT ISGYQUAAHdr, is returned followed by additional data mapped by
ISGQUERY macro

ISGYQUAARs, ISGYQUAARx, ISGYQUAARq, and ISGYQUAARqx. Note that
the ANSDetail specified determines which data is returned.

To code: Specify the RS-type address, or address in register (2)-(12), of a
character field.

,ANSAREA=ansarea
When SCANACTION=RESUME and REQINFO=QSCAN are specified, a
required output parameter, which is to contain the returned information. The
area is mapped by macro ISGYQUAA. A header area, mapped by DSECT
ISGYQUAAHdr, is returned followed by additional data mapped by
ISGYQUAARs, ISGYQUAARx, ISGYQUAARq, and ISGYQUAARqx. Note that
the ANSDetail specified determines which data is returned.

To code: Specify the RS-type address, or address in register (2)-(12), of a
character field.

,ANSDETAIL=SUMMARY, ANSDetail=FULL, ANSDetail=FULL2, ANSDetail=FULL3
When SCANACTION=START and REQINFO=QSCAN are specified, an
optional parameter that indicates the detail level of the information that should
be returned in the answer area. The default is ANSDetail=SUMMARY.

,ANSDETAIL=SUMMARY
indicates to only return ISGYQUAAHdr, ISGYQUAARs, and
ISGYQUAARq answer area data records. See ISGYQUAA mapping macro
to know what data is returned in each type of record.

,ANSDETAIL=FULL
indicates to return ISGYQUAAHdr, ISGYQUAARs, ISGYQUAARq, and
ISGYQUAARqx answer area data records. See ISGYQUAA mapping macro
to know what data is returned in each type of record.

,ANSDETAIL=FULL2
indicates that in addition to the records returned by ANSDetail=FULL,
the ISGYQUAARxs and the larger FULL2 version of the ISGYQUAARqx is
returned. See ISGYQUAA mapping macro to know what data is returned
in each type of record.

,ANSDETAIL=FULL3
indicates that in addition to the records returned by ANSDetail=FULL2,
USERDATA is returned for any records that specified USERDATA on
ISGENQ. Note that when GATHERFROM=SYSPLEX is specified and GRS
is operating in STAR mode, USERDATA is not returned for any global
requests. See ISGYQUAA mapping macro to know what data is returned
in each type of record.

,ANSLEN=anslen
When SCANACTION=START and REQINFO=QSCAN are specified, a required
input parameter that is the length of the answer area provided. The minimum
size is the amount needed to describe a single resource with a single requester.
Use an answer area length of at least 4K.
• For ANSDetail=SUMMARY, the minimum is defined by constant
  ISGYQUAA_kQSCANMinSummaryAnslen.
• For ANSDetail=FULL, the minimum is defined by constant
  ISGYQUAA_kQSCANMinFullAnslen.
• For ANSDetail=FULL2, the minimum is defined by constant
  ISGYQUAA_kQSCANMinFull2Anslen.
For ANSDETAIL=FULL3, the minimum is defined by constant ISGYQUAA_kQSCANMinFull3Anslen. The length of the answer area is at least 4k. To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

\ANSLEN=anslen

When SCANACTION=RESUME and REQINFO=QSCAN are specified, a required input parameter that is the length of the answer area provided. The minimum size is the amount needed to describe a single resource with a single requester. Use an answer area length of at least 4K. For ANSDETAIL=SUMMARY, the minimum is defined by constant ISGYQUAA_kQSCANMinSummaryAnslen. For ANSDETAIL=FULL, the minimum is defined by constant ISGYQUAA_kQSCANMinFullAnslen. For ANSDETAIL=FULL2, the minimum is defined by constant ISGYQUAA_kQSCANMinFull2Anslen. For ANSDETAIL=FULL3, the minimum is defined by constant ISGYQUAA_kQSCANMinFull3Anslen. use an answer area length of at least 4K. To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

\ANSLEN=anslen

When REQINFO=LATCHECA is specified, a required input parameter that is the length of the answer area provided. The minimum size is the amount needed to describe a single resource with a single requester. Use an answer area length of at least 4K. To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

\ANSLEN=anslen

When REQINFO=USAGESTATS is specified, a required input parameter that is the length of the answer area provided. The minimum size is the amount needed to describe the ENQ, QScan, and latch usage of a single address space as well as the usage information for terminated address spaces. The minimum is defined by constant ISGYQUAA_kUSAGESTATSMinAnslen. Use an answer area length of at least 4K. To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

\ASID=asid

When REQINFO=ENQSTATS is specified, a required input parameter that is the ASID of the address space specific information to be returned. Note that ASIDs are reusable. Once an address space has terminated another may be created with the same ASID. To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

\ASID=asid
\ASID=ANY_ASID

When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the ASID of the requesting tasks for which resource information is to be returned. Only information on requesters with that ASID is returned.
Note that ASIDs are reusable. Once an address space has terminated another may be created with the same ASID.

The default is ANY_ASID.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

\texttt{,ENQTOKEN=enqtoken}

When SEARCH\texttt{=}BY\_ENQTOKEN, SCANACTION\texttt{=}START and REQINFO\texttt{=}QSCAN are specified, a required input parameter that is the ENQToken of the request that is to be queried. Note: ENQTokens are only valid on the system where the ENQ request was made.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

\texttt{,GATHERFROM\_TYPE}\texttt{=}SYSTEM

\texttt{,GATHERFROM\_TYPE}\texttt{=}SYSPLEX

When SCANACTION\texttt{=}START and REQINFO\texttt{=}QSCAN are specified, an optional parameter that designates the extent to which the search is taken. Information about other systems is always available locally in a global resource serialization ring complex, so this keyword is ignored and forced to GATHERFROM\_TYPE\texttt{=}SYSTEM.

Use the SYSNAME keyword to obtain only information about one particular system.

Note: Only SYSTEMS scope information is obtained from other systems in the global resource serialization complex.

The default is GATHERFROM\_TYPE\texttt{=}SYSTEM.

\texttt{,GATHERFROM\_TYPE}\texttt{=}SYSTEM

Indicates to search only the caller's system. The answer area data contains information about requesters on other systems in the complex only if that information is already available on the caller's system. The returned information might be incomplete regarding requesters on other systems, including counts of the number of requesters for a resource. If performance is an issue, use GATHERFROM\_TYPE\texttt{=}SYSTEM. This request is always handled without placing the caller's dispatchable unit into a wait.

\texttt{,GATHERFROM\_TYPE}\texttt{=}SYSPLEX

Indicates to search the caller's sysplex. The answer area data contains information about requesters in the entire sysplex. If complete information regarding requesters in the sysplex is required use GATHERFROM\_TYPE\texttt{=}SYSPLEX. There are significant performance implications for this search and the caller might be suspended while the information is being gathered. Do not specify GATHERFROM\_TYPE\texttt{=}SYSPLEX if this condition cannot be tolerated.

GATHERFROM\_TYPE\texttt{=}SYSPLEX is mutually exclusive with the USERDATAMATCH\texttt{=}SPECIFIC and USERDATAMATCH\texttt{=}PATTERN filter options.

When global resource serialization is in STAR mode, GATHERFROM\_TYPE\texttt{=}SYSPLEX with ANSDETAIL\texttt{=}FULL3 results in no user data being returned for global requests.

\texttt{,JOBNAME\texttt{=}jobname}

\texttt{,JOBNAME\texttt{=}ANY\_JOBNAME}

When SEARCH\texttt{=}BY\_FILTER, SCANACTION\texttt{=}START and REQINFO\texttt{=}QSCAN
ISGQUERY macro

are specified, an optional input parameter that is the job name of the requesting tasks for which resource information is to be returned. Only information on requesters with that job name is returned. The default is ANY_JOBNAME.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

, MF=S
, MF=(L, list addr)
, MF=(L, list addr, attr)
, MF=(L, list addr, 0D)
, MF=(E, list addr)
, MF=(E, list addr, COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

, list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

, attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of 0D.

, COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

, MINOWNERS=minowners
, MINOWNERS=NO_MINOWN
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the minimum number of owners of a resource required for that resource to be returned. If any of the conditions specified by MINREQUESTERS, MINOWNERS, or MINWAITERS is met, even if the other two are not met, information for that resource and its requesters is returned. The default is NO_MINOWN.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

, MINREQUESTERS=minrequesters
, MINREQUESTERS=NO_MINREQ
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN
are specified, an optional input parameter that is the minimum number of owners plus waiters for a resource required for that resource to be returned. If any of the conditions specified by MINREQUESTERS, MINOWNERS, or MINWAITERS is met, even if the other two are not met, information for that resource and its requesters is returned. The default is NO_MINREQ.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

\texttt{MINWAITERS} = minwaiters
\texttt{MINWAITERS} = NO_MINWAIT

When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the minimum number of waiters for a resource required for that resource to be returned. If any of the conditions specified by MINREQUESTERS, MINOWNERS, or MINWAITERS is met, even if the other two are not met, information for that resource and its requesters is returned. The default is NO_MINWAIT.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

\texttt{PLISTVER} = IMPLIED_VERSION
\texttt{PLISTVER} = MAX
\texttt{PLISTVER} = 1
\texttt{PLISTVER} = 2

An optional input parameter in the 1-2 range that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **1**, if you use the currently available parameters:
ISGQUERY macro

- QNAMEMATCH
- REQINFO
- REQUESTERLIMIT
- RESUMETOKEN
- RNAME
- RNAMELEN
- RNAMEMATCH
- RNL
- RNLE
- SCANACTION
- SCOPE
- SEARCH
- SERIALIZEBY
- SYSNAME
- TTOKEN
- 2, which supports both the following parameters and those from version 1:
  - USERDATA
  - USERDATALEN
  - USERDATAMATCH

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 1, or 2

,QNAME=qname
When REQINFO=RNLSEARCH is specified, a required input parameter that is the QName of the resource for which the RNLs are to be searched.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,QNAME=qname
When QNAMEMATCH=SPECIFIC, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is the specific QName of the resources to be returned.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,QNAME=qname
When QNAMEMATCH=PATTERN, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is a pattern QName to match the resources to be returned.

The QName pattern is 8 characters where ? matches any single character, and * matches any string of zero or more characters. Note: All trailing blanks are ignored when matching QNames to QName patterns.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,QNAMEMATCH=SPECIFIC
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required parameter.

QNAMEMATCH=SPECIFIC
Indicates to only return information on resources that exactly match the specified specific QName.

QNAMEMATCH=PATTERN
Indicates to only return information on resources that match the specified QName pattern.

REQINFO=RNLSEARCH
Indicates to search a specific RNL for a given resource name.

The CMSEQDQ lock serializes the use of the RNLs, so holding this lock ensures that the RNL does not change and therefore the returned RNLE is valid on the current RNLs.

During an RNL change, the currently active RNLs are searched.

For more information about how a resource can be changed by the system, see the TEST=YES function in Chapter 114, “ISGENQ — Global resource serialization ENQ service,” on page 1131.

REQINFO=ENQSTATS
Indicates to return information related to ENQ counts.

REQINFO=QSCAN
Indicates to search the global resource serialization queues for resource and requester information.

REQINFO=LATCHECA
Indicates to search the global resource serialization queues for query latch enhanced contention analysis (ECA) data for waiters that might indicate contention issues.

Note: The LATCHECA search does not return data for blockers or dependency data.

REQINFO=USAGESTATS
Indicates to search the global resource serialization queues for address space level contention information related to ENQs (all scopes) and latches (all latch sets). Global resource serialization gathers latch statistics in requester and latch set owner address space categories. The statistics are provided for all address spaces as follows:

- **ENQ by scope:** this includes contention counts, total delay times, and the sum of the squared delay (SUMSQ) times. The SUMSQ times can be used to compute the standard deviation.

- **Latch:** For both requesters and latch set owners, this includes contention counts, total delay times, and the sum of the squared delay (SUMSQ) times

- **ENQ usage counts.** Note that latch counts are kept in “fast counts” in latch sets and not on an address space basis.
When SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the maximum number of requesters (owners and waiters) to be returned for each individual resource. Only resource related information is returned if 0 is specified. The value range of RequesterLimit is 0 to 2^15-1 (32767). The default is 32767.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

When SCANACTION=START and REQINFO=QSCAN are specified, an optional output parameter that is the resume token for this search. When RESUMETOKEN is specified, a reason code of ISGQUERYRsn_AnsweAreaFull indicates that the token can be used to resume the scan on a subsequent call. If the return code indicates that the search can be resumed, a SCANACTION=RESUME or SCANACTION=QUIT with the returned resume token must be subsequently issued.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

When SCANACTION=RESUME and REQINFO=QSCAN are specified, a required input/output parameter that is the resume token from a previously started search. If the search does not complete the resume token can be used to resume the search on a subsequent call. Check the return code to determine if the resume token can be used to resume the scan. If the return code indicates that the search can be resumed, a SCANACTION=RESUME or SCANACTION=QUIT with the returned resume token must be subsequently issued.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

When SCANACTION=QUIT and REQINFO=QSCAN are specified, a required input/output parameter that is the resume token from a previously started search. Any global resource serialization storage associated with the search is freed, and the resume token is cleared to binary zeros.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

**To code:** Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

When REQINFO=RNLSSEARCH is specified, a required input parameter that is the RName of the resource for which the RNLS are to be searched.

The RName pattern is a string of characters where ? matches any single character, and * matches any string of zero or more characters.

**To code:** Specify the RS-type address, or address in register (2)-(12), of a character field.
ISGQUERY macro

\texttt{RNAME}=	extit{rname}

When \texttt{RNAMEMATCH=SPECIFIC}, \texttt{SEARCH=BY\_FILTER}, \texttt{SCANACTION=START} and \texttt{REQINFO=QSCAN} are specified, a required input parameter that is the specific RName of the resources to be returned.

\textbf{To code}: Specify the RS-type address, or address in register (2)-(12), of a character field.

\texttt{RNAME}=	extit{rname}

When \texttt{RNAMEMATCH=PATTERN}, \texttt{SEARCH=BY\_FILTER}, \texttt{SCANACTION=START} and \texttt{REQINFO=QSCAN} are specified, a required input parameter that is a pattern RName to match the resources to be returned. The RName pattern is a string of characters where '?' matches any single character, and '*' matches any string of zero or more characters.

\textbf{To code}: Specify the RS-type address, or address in register (2)-(12), of a character field.

\texttt{RNAMELEN}=	extit{rnamelen}

When \texttt{REQINFO=RNLSEARCH} is specified, a required input parameter that is the length of the given RName. The specified length can be 1 to 255.

\textbf{To code}: Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

\texttt{RNAMELEN}=	extit{rnamelen}

When \texttt{RNAMEMATCH=SPECIFIC}, \texttt{SEARCH=BY\_FILTER}, \texttt{SCANACTION=START} and \texttt{REQINFO=QSCAN} are specified, a required input parameter that is the length of the given RName. The specified length can be 1 to 255.

\textbf{To code}: Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

\texttt{RNAMELEN}=	extit{rnamelen}

When \texttt{RNAMEMATCH=PATTERN}, \texttt{SEARCH=BY\_FILTER}, \texttt{SCANACTION=START} and \texttt{REQINFO=QSCAN} are specified, a required input parameter that is the length of the given RName pattern. The specified length can be 1 to 255.

\textbf{To code}: Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

\texttt{RNAMEMATCH=ANY}
\texttt{RNAMEMATCH=SPECIFIC}
\texttt{RNAMEMATCH=PATTERN}

When \texttt{SEARCH=BY\_FILTER}, \texttt{SCANACTION=START} and \texttt{REQINFO=QSCAN} are specified, a required parameter.

\texttt{RNAMEMATCH=ANY}

Indicates to return information on resources with any RName.

\texttt{RNAMEMATCH=SPECIFIC}

Indicates to only return information on resources that exactly match the specified specific RName.

\texttt{RNAMEMATCH=PATTERN}

Indicates to only return information on resources that match the specified RName pattern.

\texttt{RNL=SIRNL}
\texttt{RNL=SERNL}
ISQUERY macro

\[ \text{RNL=RCLNL} \]
When REQINFO=RNLSEARCH is specified, a required parameter that indicates which resource name list (RNL) is to be searched.

\[ \text{RNL=SIRNL} \]
Indicates to search the system inclusion RNL.

\[ \text{RNL=SERNL} \]
Indicates to search the systems exclusion RNL.

\[ \text{RNL=RCRNL} \]
Indicates to search the reserve conversion RNL.

\[ \text{RNLE=rnle} \]
When REQINFO=RNLSEARCH is specified, an optional output parameter that is a copy of the matching RNLE. The caller must include the ISGRNLE macro to get a mapping for the RNLE.

Note: The RNLE returned is dependent on the version of the parameter list. If a new version of the RNLE should be introduced, it might require a larger character field. Explicitly state the PLISTVER to ensure that the size of the RNLE returned does not change.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

\[ \text{RSNCODE=rsncode} \]
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

\[ \text{SCANACTION=START} \]
\[ \text{SCANACTION=RESUME} \]
\[ \text{SCANACTION=QUIT} \]
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required parameter that designates whether to start, resume, or quit a QScan.

\[ \text{SCANACTION=START} \]
Indicates to start a search of the global resource serialization queues.

\[ \text{SCANACTION=RESUME} \]
Indicates to resume a previously started search.

\[ \text{SCANACTION=QUIT} \]
Indicates to quit a previously started search. If a started search has not completed it must be either resumed until it completes or ended with SCANACTION=QUIT.

\[ \text{SCOPE=ANY} \]
\[ \text{SCOPE=STEP} \]
\[ \text{SCOPE=SYSTEM} \]
\[ \text{SCOPE=SYSTEMS} \]
\[ \text{SCOPE=SYSPLEX} \]
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional parameter that is the scope of the resources to be returned.

Note: Only information on resources with scope of SYSTEMS is returned from systems other than the caller’s system.
The default is SCOPE=ANY.

**SCOPE=ANY**
Indicates to return information on resources with any scope.

**SCOPE=STEP**
Indicates to only return information on resources with a scope of STEP.

**SCOPE=SYSTEM**
Indicates to only return information on resources with a scope of SYSTEM.

**SCOPE=SYSTEMS**
Indicates to only return information on resources with a scope of SYSTEMS or SYSPLEX.

**SCOPE=SYSPLEX**
Indicates to only return information on resources with a scope of SYSTEMS or SYSPLEX. (SYSPLEX is an alias for SYSTEMS.)

**SEARCH=BY_ENQTOKEN**
**SEARCH=BY_FILTER**
When SCANACT=START and REQINFO=QSCAN are specified, a required parameter that designates the method to search for resources.

**SEARCH=BY_ENQTOKEN**
Indicates to search using a specific ENQToken. Information is returned about the requester of the ENQ and the resource for which the ENQ was requested.

**SEARCH=BY_FILTER**
Indicates to search on resource and requester characteristics using filters. Information is returned about the resources and requesters that match the search criteria.

**SERIALIZEBY=ANY**
**SERIALIZEBY=RESERVE**
**SERIALIZEBY=ENQ_ONLY**
When SEARCH=BY_FILTER, SCANACT=START and REQINFO=QSCAN are specified, an optional parameter that indicates if information should be returned depending on whether the requests are serialized by device reserves. The default is SERIALIZEBY=ANY.

**SERIALIZEBY=ANY**
Indicates to return information on requests of any type.

**SERIALIZEBY=RESERVE**
Indicates to only return information on reserve requests that were not converted.

**SERIALIZEBY=ENQ_ONLY**
Indicates to only return information on requests that do not result in a device reserve. This includes reserve requests that were converted to global ENQs. Answer area bit ISGYQUAARqReserveConverted is set for reserve requests that were converted.

**SYSNAME=sysname**
**SYSNAME=ANY_SYSNAME**
When SEARCH=BY_FILTER, SCANACT=START and REQINFO=QSCAN are specified, an optional input parameter that is the system name of the requesting tasks for which resource information is to be returned. Only information on requesters in that system is returned. If
ISGQUERY macro

GATHERFROM=SYSTEM is specified (or is the default), SYSNAME might only be the name of the caller's system or the default of ANY_SYSNAME.

Note: Only information on resources with scope of SYSTEMS is returned from systems other than the caller's system.

The default is ANY_SYSNAME.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,TTOKEN=ttoken
,TTOKEN=ANY_TTOKEN

When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the task token of the requesting task for which resource information is to be returned. Only information on that requester is returned. The TToken specified is valid only on the current system.

Note: The TToken of requesters is unavailable for ENQs obtained before the global resource serialization address space was created. The TToken filter will not match those ENQ requesters.

The default is ANY_TTOKEN.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,USERDATA=userdata

When USERDATAMATCH=SPECIFIC, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is the specific UserData of the requests to be returned.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,USERDATA=userdata

When USERDATAMATCH=_PATTERN, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is a pattern UserData to match the requests to be returned. The UserData pattern is a string of characters where '?' matches any single character, and '*' matches any string of zero or more characters.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,USERDATALEN=userdatalen

When USERDATAMATCH=_PATTERN, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is the length of the given UserData pattern. The specified length can be 1 to 32.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

,USERDATAMATCH=ANY
,USERDATAMATCH=SPECIFIC
,USERDATAMATCH=_PATTERN

When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional parameter that indicates which requests to return. The default is USERDATAMATCH=ANY.
ISGQUERY macro

,USERDATAMATCH=ANY
indicates to return information on request with any USERDATA, including those with no USERDATA.

,USERDATAMATCH=SPECIFIC
indicates to only return requests that have USERDATA that exactly matches the specified USERDATA. For information about specifying USERDATA on an ISGENQ request, see Chapter 114, “ISGENQ — Global resource serialization ENQ service,” on page 1131. Note that USERDATA can only be attached to a request through the ISGENQ interface.

This request requires a version 2 parameter list.

GATHERFROM=SYSPLEX is mutually exclusive with the USERDATAMATCH=SPECIFIC option.

,USERDATAMATCH=PATTERN
indicates to only return information on requests that match the specified UserData pattern. For information about specifying USERDATA on an ISGENQ request, see Chapter 114, “ISGENQ — Global resource serialization ENQ service,” on page 1131.

All trailing blanks are not ignored when matching USERDATA to USERDATA patterns. For example, if the USERDATA is ABC123, and the pattern used to search is A*3, it does not match. A pattern such as A*3* does match.

Note: Userdata can only be attached to a request through the ISGENQ interface.

This request requires a version 2 parameter list.

GATHERFROM=SYSPLEX is mutually exclusive with the USERDATAMATCH=PATTERN option.

ABEND codes
None.

Return and reason codes
When the ISGQUERY macro returns control to your program:
- GPR 15 (and retcode, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and rsncode, when you code RSNCODE) contains a reason code.

Macro ISGYCON provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the xxxx value.
### ISGQUERY macro

**Table 113. Return and Reason Codes for the ISGQUERY Macro**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>—</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRc_OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: ISGQUERY request successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For REQINFO=RNLSEARCH, a matching RNLE was found for the given resource name. For REQINFO=QSCAN, processing complete and data has been copied into the answer area. There is no more data to return.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: None required.</td>
</tr>
<tr>
<td>04</td>
<td>—</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRc_Warn</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: Warning. ISGQUERY completed successfully, however a warning has been issued.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Refer to action under the individual reason code.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0401</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRsn_NoMatchingRNLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: For a REQINFO=RNLSEARCH request. No matching RNLE was found for the given resource name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: No action required.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0402</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRsn_RNLChangeInProgess</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: For a REQINFO=RNLSEARCH request. A matching RNLE was found for the given resource name, but an RNL change is in progress in the system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: No action required.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0403</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRsn_GRSRNLExclude</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: For a REQINFO=RNLSEARCH request. GRSRNL=EXCLUDE is in effect. When GRSRNL=EXCLUDE the RNLS are not used and all SYSTEMS scope requests are forced to SYSTEM. An alternative serialization product may be in use. No RNLE is returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: No action required.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0404</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRsn_NoMatchingResources</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: For REQINFO=QSCAN and REQINFO=LatchECA requests. While scanning the queues, no resources were found that match the caller’s request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: No action required.</td>
</tr>
</tbody>
</table>
### Table 113. Return and Reason Codes for the ISGQUERY Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>xxxx0405</td>
<td></td>
<td>Equate Symbol: ISGQUERYRsn_AnswerAreaFull. Meaning: For a REQINFO=QSCAN request, ISGQUERY has provided some data, however the answer area is too small to contain all the requested data. Action: The user should process the data in the answer area. If RESUMETOKEN was not specified on the request and more information is needed, re-issue the request with a larger answer area or specify a resume token. If RESUMETOKEN was specified, either issue a REQINFO=QSCAN SCANACTION=RESUME request with the returned resume token to continue the scan, or issue REQINFO=QSCAN SCANACTION=QUIT to end the search.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0406</td>
<td></td>
<td>Equate Symbol: ISGQUERYRsn_GRSNone. Meaning: For a REQINFO=RNLSEARCH request, GRS=NONE is in effect. When GRS=NONE the RNLs are not used and all requests are serialized only within the current system. Note that though both scope SYSTEM and SYSTEMS requests are local to the current system, they still represent separate resources and are NOT serialized with each other.</td>
</tr>
<tr>
<td>08</td>
<td>—</td>
<td></td>
<td>Equate Symbol: ISGQUERYRc_ParmError. Meaning: ISGQUERY request specified parameters in error. Action: Refer to action under the individual reason code.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0801</td>
<td></td>
<td>Equate Symbol: ISGQUERYRsn_BadPlistAddress. Meaning: Unable to access parameter list. Action: Check that the entire parameter list is addressable. If in AR-mode, check that the ALET of the parameter list is correct. Note that if this macro is issued in AR-mode, SYSSTATE ASCENV=AR must be issued before this macro. Ensure that the storage is in the same key as the caller.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0802</td>
<td></td>
<td>Equate Symbol: ISGQUERYRsn_BadPlistALET. Meaning: Bad parameter list ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's dispatchable unit access list (DU-AL), nor a valid entry for a common area data space. Action: Ensure that the ALET of the parameter list is valid. Its access register might have been set up properly.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0803</td>
<td></td>
<td>Equate Symbol: ISGQUERYRsn_BadPlistVersion. Meaning: Bad parameter list version number. The service level of GRS on which the caller is running does not support this version of the ISGQUERY service, or the ISGQUERY parameter list version is lower than the minimum required for parameters that were specified. Action: Check that the request has the correct version number. Check for possible storage overlay of the parameter list.</td>
</tr>
</tbody>
</table>
### Table 113. Return and Reason Codes for the ISGQUERY Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>xxxx0804</td>
<td>Equate Symbol: ISGQUERYRsnReservedFieldNotNull</td>
<td>A reserved field in the parameter list is non-zero.</td>
<td>Check for possible storage overlay of the parameter list.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0805</td>
<td>Equate Symbol: ISGQUERYRsn_BadReqInfo</td>
<td>Bad REQINFO parameter.</td>
<td>Check for possible storage overlay of the parameter list.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0806</td>
<td>Equate Symbol: ISGQUERYRsn_BadRNL</td>
<td>Bad RNL parameter.</td>
<td>Check for possible storage overlay of the parameter list.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0807</td>
<td>Equate Symbol: ISGQUERYRsn_BadRNameAddress</td>
<td>Unable to access the RName.</td>
<td>Ensure that the entire RName field is addressable. If in AR-mode, this field is accessed through its address and ALET, check that both these values are correct. Check that specified RName length is correct. Ensure that the storage is in the same key as the caller.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0808</td>
<td>Equate Symbol: ISGQUERYRsn_BadRNameALET</td>
<td>Bad RName ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's dispatchable unit access list (DU-AL), nor a valid entry for a common area data space.</td>
<td>Ensure that the ALET of the RName is valid. Its access register might have been set up properly.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0809</td>
<td>Equate Symbol: ISGQUERYRsn_BadRNameLen</td>
<td>The RName length specified is not valid.</td>
<td>Ensure the RName length field contains a number from 1-255.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx080A</td>
<td>Equate Symbol: ISGQUERYRsn_BadRNLEAddress</td>
<td>Unable to access RNLE output field.</td>
<td>Ensure that the entire RNLE field is addressable. If in AR-mode, this field is accessed through its address and ALET, check that both these values are correct. Check that RNLE length is correct. Ensure that the storage is in the same key as the caller.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx080B</td>
<td>Equate Symbol: ISGQUERYRsn_BadRNLEALET</td>
<td>Bad RNLE ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's dispatchable unit access list (DU-AL), nor a valid entry for a common area data space.</td>
<td>Ensure that the ALET of the RNLE is valid. Its access register might have been set up properly.</td>
</tr>
</tbody>
</table>
### ISGQUERY macro

Table 113. Return and Reason Codes for the ISGQUERY Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx080C    | Equate Symbol: ISGQUERYRsn_MutuallyExclusive | Meaning: Mutually exclusive keywords were specified.  
Action: Check for a possible storage overlay of the parameter list. |
| 08          | xxxx080D    | Equate Symbol: ISGQUERYRsn_BadAnsAreaAddress | Meaning: Unable to access the answer area.  
Action: Ensure that the entire answer area is addressable. If in AR-mode, this field is accessed through its address and ALET, check that both these values are correct. Check that the specified answer area length is correct. Ensure that the storage is in the same key as the caller. |
| 08          | xxxx080E    | Equate Symbol: ISGQUERYRsn_BadAnsAreaALET | Meaning: Bad answer area ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.  
Action: Ensure that the ALET of the answer area is valid. Its access register might have been set up properly. |
| 08          | xxxx080F    | Equate Symbol: ISGQUERYRsn_BadScanAction | Meaning: Bad SCANACTION parameter.  
Action: Check for possible storage overlay of the parameter list. |
| 08          | xxxx0810    | Equate Symbol: ISGQUERYRsn_BadResumeTokenAddress | Meaning: Unable to access the ResumeToken.  
Action: Ensure that the entire ResumeToken is addressable. If in AR-mode, this field is accessed through its address and ALET, check that both these values are correct. Ensure that the storage is in the same key as the caller. |
| 08          | xxxx0811    | Equate Symbol: ISGQUERYRsn_BadResumeTokenALET | Meaning: Bad ResumeToken ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller’s dispatchable unit access list (DU-AL), nor a valid entry for a common area data space.  
Action: Ensure that the ALET of the ResumeToken is valid. Its access register might not have been set up properly. |
| 08          | xxxx0812    | Equate Symbol: ISGQUERYRsn_BadGatherFrom | Meaning: Bad GATHERFROM parameter.  
Action: Check for possible storage overlay of the parameter list. |
| 08          | xxxx0813    | Equate Symbol: ISGQUERYRsn_BadSearch | Meaning: Bad SEARCH keyword parameter.  
Action: Check for possible storage overlay of the parameter list. |
### Table 113. Return and Reason Codes for the ISGQUERY Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code little-endian encoded value</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0814                               | **Equate Symbol**: ISGQUERYRsn_BadENQTokenAddress  
**Meaning**: Unable to access the ENQToken.  
**Action**: Ensure that the entire ENQToken is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Ensure that the storage is in the same key as the caller. |
| 08          | xxxx0815                               | **Equate Symbol**: ISGQUERYRsn_BadENQTokenALET  
**Meaning**: Bad ENQToken ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller’s dispatchable unit access list (DU-AL), nor a valid entry for a common area data space.  
**Action**: Ensure that the ALET of the ENQToken is valid. Its access register might have been set up properly. |
| 08          | xxxx0816                               | **Equate Symbol**: ISGQUERYRsn_BadQNameMatch  
**Meaning**: Bad QNAMEMATCH keyword parameter.  
**Action**: Check for possible storage overlay of the parameter list. |
| 08          | xxxx0817                               | **Equate Symbol**: ISGQUERYRsn_BadRNameMatch  
**Meaning**: Bad RNAMEMATCH keyword parameter.  
**Action**: Check for possible storage overlay of the parameter list. |
| 08          | xxxx0818                               | **Equate Symbol**: ISGQUERYRsn_BadScope  
**Meaning**: Bad SCOPE keyword parameter.  
**Action**: Check for possible storage overlay of the parameter list. |
| 08          | xxxx0819                               | **Equate Symbol**: ISGQUERYRsn_BadSerializeBy  
**Meaning**: Bad SERIALIZEBY keyword parameter.  
**Action**: Check for possible storage overlay of the parameter list. |
| 08          | xxxx081A                               | **Equate Symbol**: ISGQUERYRsn_AnsLenTooSmall  
**Meaning**: The size of the answer area is not large enough to contain the minimal amount of information.  
**Action**: Increase the answer area size to at least the minimum required for the specified request. See the provided constants. However, the answer area length should be at least 4k. |
| 08          | xxxx081B                               | **Equate Symbol**: ISGQUERYRsn_ResumeTokenNotValid  
**Meaning**: The specified resume token is not a valid resume token.  
**Action**: Ensure the resume token is from a previously started search on the current system. |
| 08          | xxxx081C                               | **Equate Symbol**: ISGQUERYRsn_ResumeTokenTooOld  
**Meaning**: The specified resume token is from an old search request that has expired.  
**Action**: Restart the search if more information is needed. |
### Table 113. Return and Reason Codes for the ISGQUERY Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>xxxx081D</td>
<td>Equate Symbol: ISGQUERYRsn_ENQTokenNotValid</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: The ENQToken specified is not a valid ENQToken.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Ensure the ENQToken is from a previous ISGENQ request on the current system.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx081E</td>
<td>Equate Symbol: ISGQUERYRsn_BadRequesterLimit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: The REQUESTERLIMIT value specified is not valid. RequesterLimit must be 0 to 275-1 (32767).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Ensure that the requester limit is in the correct range.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx081F</td>
<td>Equate Symbol: ISGQUERYRsn_NoPossibleMatch</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: For a REQINFO=QSCAN request. Conflicting parameters were specified such that no resources could possibly match the request. A SYSNAME other than the current system was specified along with SCOPE=STEP, SCOPE=SYSTEM, TTOKEN, or GATHERFROM=SYSTEM. Or SERIALIZEBY=RESERVE was specified with SCOPE=STEP.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Avoid specifying conflicting parameters.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx0820</td>
<td>Equate Symbol: ISGQUERYRsn_BadAnsDetail</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: Bad ANSDETAIL keyword parameter.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check for possible storage overlay of the parameter list.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx0821</td>
<td>Equate Symbol: ISGQUERYRsn_NotAuthToQscan</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: SETROPTS MLACTIVE is in effect, and the program is not authorized to issue ISGQUERY REQINFO=QSCAN.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Ensure the program is running authorized, or is associated with a userid with at least READ access to the best fit FACILITY class resource profile of the form ISG,QSCAN,SERVICES,AUTHORIZATION and that the FACILITY class is SETROPTS RACLISTed.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx0822</td>
<td>Equate Symbol: ISGQUERYRsn_BadASID</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: Bad ASID keyword parameter.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Ensure that the ASID is valid.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx0823</td>
<td>Equate Symbol: ISGQUERYRsn_BadUserDataAddress</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meaning: Unable to access the userdata.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Ensure that the entire USERDATA is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. If this is a USERDATA pattern request, check that specified USERDATA length is correct. Ensure that the storage is in the same key as the caller.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 113. Return and Reason Codes for the ISGQUERY Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>xxxx0824</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRsn_BadUserDataAlet</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: Bad USERDATA ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space.</td>
<td><strong>Action</strong>: Ensure that the ALET of the USERDATA is valid. Its access register might have been set up properly.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0825</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRsn_BadUserDataLen</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: The USERDATA length specified is not valid.</td>
<td><strong>Action</strong>: Ensure the USERDATA length field contains a number in the range 1-32.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0826</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRsn_BadUserDataMatch</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: Bad USERDATAMATCH keyword parameter.</td>
<td><strong>Action</strong>: Check for possible storage overlay of the parameter list.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0827</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRsn_BadAnalyze</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: The ANALYZE keyword parameter is not valid.</td>
<td><strong>Action</strong>: Check for possible storage overlay of the parameter list.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0828</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRsn_NotAuthToLatchECA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: SETROPTS MLACTIVE is in effect and the program is not authorized to issue ISGQUERY REQINFO=LATCHECA.</td>
<td><strong>Action</strong>: Ensure the program is running authorized or is associated with a userid with at least READ access to the best fit FACILITY class resource profile of the form ISG.LATCHECASVCS.AUTHORIZATION and that the FACILITY class is SETROPTS RAACLSTed.</td>
</tr>
<tr>
<td>0C</td>
<td>—</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRc_EnvError</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: ISGQUERY request has an environment error.</td>
<td><strong>Action</strong>: Refer to action under the individual reason code.</td>
</tr>
<tr>
<td>0C</td>
<td>xxxx0C01</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRsn_SrbMode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: ISGQUERY can not be used in SRB mode.</td>
<td><strong>Action</strong>: Avoid using ISGQUERY in SRB mode.</td>
</tr>
<tr>
<td>0C</td>
<td>xxxx0C02</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRsn_NotEnabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: ISGQUERY can not be used disabled.</td>
<td><strong>Action</strong>: Avoid using ISGQUERY when not enabled.</td>
</tr>
<tr>
<td>0C</td>
<td>xxxx0C03</td>
<td><strong>Equate Symbol</strong>: ISGQUERYRsn_ComplexMigrating</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning</strong>: For a REQINFO=QSCAN request. The ISGQUERY service failed because the GRS complex was migrating from a ring to a star configuration.</td>
<td><strong>Action</strong>: Retry the request on or more times.</td>
</tr>
</tbody>
</table>
### Table 113. Return and Reason Codes for the ISGQUERY Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0C          | xxxx0C04    | **Equate Symbol:** ISGQUERYRsn_CannotObtainLocks  
**Meaning:** For REQINFO=RNLSEARCH, the local and CMSEQDQ locks could not be obtained.  
**Action:** Only use ISGQUERY REQINFO=RNLSEARCH when either no locks are held, or both a local lock and the CMSEQDQ lock are held with no other locks. |
| 0C          | xxxx0C05    | **Equate Symbol:** ISGQUERYRsn_LockHeld  
**Meaning:** An incorrect lock was held upon entry. For REQINFO=QSCAN, no locks may be held. For REQINFO=RNLSEARCH, either no locks or both a local lock (LOCAL or CML) and the CMDEQDQ lock must be held.  
**Action:** Avoid using ISGQUERY REQINFO=QSCAN when locks are held. Avoid using ISGQUERY REQINFO=RNLSEARCH when locks other than both a local lock and the CMSEQDQ lock are held. |
| 0C          | xxxx0C06    | **Equate Symbol:** ISGQUERYRsn_MaxConcurrentRequests  
**Meaning:** For a REQINFO=QSCAN request. The answer area was filled before queue scan processing completed, and reason code ISGQUERYRsn_AnswerAreaFull would have been issued. However, RESUMETOKEN was specified, but the limit for the number of concurrent resource requests (ISGENQ, ENQ, RESERVE, GQSCAN, and ISGQUERY) has been reached. The data in the answer area is valid, but incomplete. The scan cannot be resumed.  
**Action:** Retry the request one or more times. If the problem persists, consult your system programmer. For more information on concurrent count limits and how the system can be tuned when necessary, see [z/OS MVS Planning: Global Resource Serialization](https://pubs咪咪m.9/213/1277/01277_mv5_pla_glo_ser.shtm). |
| 0C          | xxxx0C07    | **Equate Symbol:** ISGQUERYRsn_RingResumeInStar  
**Meaning:** For a REQINFO=QSCAN request. The caller attempted to resume a scan that was started when the global resource serialization complex, which is now in star mode, was in ring mode.  
**Action:** Reissue the original request. |
| 0C          | xxxx0C08    | **Equate Symbol:** ISGQUERYRsn_InsufficientStorage  
**Meaning:** For a REQINFO=QSCAN request. The ISGQUERY service could not obtain storage to satisfy the request.  
**Action:** Retry the request one or more times. |
| 0C          | xxxx0C09    | **Equate Symbol:** ISGQUERYRsn_FRRHeld,  
**Meaning:** For a REQINFO=LATCHeca request. The caller issued ISGQUERY with a functional recover routine (FRR) established.  
**Action:** Avoid issuing ISGQUERY REQINFO=LATCHeca when using functional recovery routines. |
### Table 113. Return and Reason Codes for the ISGQUERY Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
</table>
| 10          | —           | Equate Symbol: ISGQUERYRe_CompError  
**Meaning:** Component Error  
**Action:** Contact the IBM Support Center.  
The reason code contains internal diagnostic information. |

### Examples

Use these examples as a guide.

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* Search the Systems Inclusion RNL for a resource name
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```
ISGQUERY REQINFO=RNLSEARCH,RNL=SIRNL, X
   QNAME=MYQNAME,RNAME=MYRNAME,RNAMELEN=MYNAMELEN, X
   RETCODE=MYRC,RSNCODE=MYRSN
```

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* Query information on a request specified by ENQToken
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```
ISGQUERY REQINFO=QSCAN,SCANACTION=START, X
   ANSCAREA=MYAREA,ANSLEN=MYAREALEN, X
   SEARCH=BY_ENQTOKEN,ENQTOKEN=MYENQTOKEN, X
   RETCODE=MYRC,RSNCODE=MYRSN
```

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* Start a resumable query for resources of a specific job that
* matches a specific QNAME and pattern RNAME
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```
ISGQUERY REQINFO=QSCAN,SCANACTION=START, X
   ANSCAREA=MYAREA,ANSLEN=MYAREALEN, X
   SEARCH=BY_FILTER,QNAMEMATCH=SPECIFIC,QNAME=MYQNAME, X
   RNAMEMATCH=_PATTERN,RNAME=CL7'ABC?23*',RNAMELEN=7, X
   USERDATAMATCH=SPECIFIC,USERDATA=MYUDATA, X
   JOBNAME=MYJOBNAME,RESUMETOKEN=MYRESTOKEN,RETCODE=MYRC, X
   RSNCODE=MYRSN
```

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* Start a resumable query for resources of a specific job that
* matches a specific QNAME and pattern RNAME
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```
ISGQUERY REQINFO=QSCAN,SCANACTION=START, X
   ANSCAREA=MYAREA,ANSLEN=MYAREALEN, X
   SEARCH=BY_FILTER,QNAMEMATCH=SPECIFIC,QNAME=MYQNAME, X
   RNAMEMATCH=_PATTERN,RNAME=CL7'ABC?23*',RNAMELEN=7, X
   USERDATAMATCH=SPECIFIC,USERDATA=MYUDATA, X
   JOBNAME=MYJOBNAME,RESUMETOKEN=MYRESTOKEN,RETCODE=MYRC, X
   RSNCODE=MYRSN
```

*MYUDATA DC CL32'MY USERDATA'
MYUDATAP DC CL7'M???USE*'  

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* Resume a query that was started but not completed
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```
ISGQUERY REQINFO=QSCAN,SCANACTION=RESUME, X
```

z/OS V2R1.0 MVS Authorized Assembler Services Reference EDT-DXG
ISGQUERY macro

RESUMETOKEN=MYRESTOKEN, X
ANSAREA=MYAREA,ANSLEN=MYAREALEN, X
RETCODE=MYRC,RSNCODE=MYRSN

* *************************************************************
* Quit a query that was started but not completed
* *************************************************************

ISGQUERY REQINFO=QSCAN,SCANACTION=QUIT, X
RESUMETOKEN=MYRESTOKEN, X
RETCODE=MYRC,RSNCODE=MYRSN

* *************************************************************
* Gather ENQ statistics for a particular address space
* *************************************************************

ISGQUERY REQINFO=ENQSTATS, X
ANSAREA=MYAREA,ASID=MYASID, X
RETCODE=MYRC,RSNCODE=MYRSN

* *************************************************************
* Gather query latch enhanced contention analysis (LATCHECA) data from the *
* global resource serialization queues for waiters delayed because of *
* contention
* *************************************************************

ISGQUERY REQINFO=LATCHECA,ANALYZE=WAITER,ANSAREA=MYAREA, X
ANSLEN=MYAREALEN,RETCODE=MYRC,RSNCODE=MYRSN X

* *************************************************************
* Gather address space level contention information related to ENQs *
* (all scopes) and latches (all latch sets) from the *
* global resource serialization queues *
* *************************************************************

ISGQUERY REQINFO=USAGESTATS,ANSAREA=MYAREA,ANSLEN=MYAREALEN, X
RETCODE=MYRC,RSNCODE=MYRSN X

For more information on global resource serialization, see z/OS MVS Planning.

Global Resource Serialization
Chapter 128. ITTFMTB — Generate component trace format table

Description

ITTFMTB generates a table called the component trace format table. It can also generate a map of the table. IPCS uses this table to control the formatting of trace data for program events that occur when the system runs. When you use ITTFMTB to generate information in the table, you are specifying the formatting style of the trace data. For information about IPCS, see z/OS MVS IPCS User’s Guide and z/OS MVS IPCS Customization.

Invoke the macro once to define the beginning of the table and once to define the end of the table. In between, you can invoke the macro repeatedly to define the individual formats for the various traceable events.

This macro generates nonexecutable code, and therefore is not sensitive to the execution environment.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PSW key 0</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN or PASN~=HASN~=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>24- or 31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt Status</td>
<td>Enabled or disabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

None.

Register information

This macro does not use any registers.

Performance implications

None.

Syntax

The ITTFMTB macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## ITTFMTB macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: Symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td><code>bslash</code></td>
<td>One or more blanks must precede ITTFMTB.</td>
</tr>
<tr>
<td></td>
<td>ITTFMTB</td>
</tr>
<tr>
<td><code>bslash</code></td>
<td>One or more blanks must follow ITTFMTB.</td>
</tr>
<tr>
<td>MAP</td>
<td>Required choice. Select one of four options.</td>
</tr>
<tr>
<td><code>TABLEDATA=</code></td>
<td><code>tabname</code>: Symbol up to eight characters long.</td>
</tr>
<tr>
<td><code>EVENTDATA=</code></td>
<td><code>eventid</code>: A-type address.</td>
</tr>
<tr>
<td><code>TABLEEND</code></td>
<td></td>
</tr>
<tr>
<td><code>,ENTRYLENGTH=</code></td>
<td><code>elength</code>: A-type address.</td>
</tr>
<tr>
<td><code>,LOCBUFNAME=</code></td>
<td><code>bufname</code>: Symbol up to eight characters long.</td>
</tr>
<tr>
<td><code>,LOCBUFADDR=</code></td>
<td><code>bufaddr</code>: A-type address.</td>
</tr>
<tr>
<td><code>,FILTERNAME=</code></td>
<td><code>pgmname</code>: Symbol up to eight characters long.</td>
</tr>
<tr>
<td><code>,FILTERADDR=</code></td>
<td><code>pgmaddr</code>: A-type address.</td>
</tr>
<tr>
<td><code>,MNEMONIC=</code></td>
<td><code>mnemonic</code>: Symbol up to 32 characters long.</td>
</tr>
<tr>
<td><code>,DESCRIPTION=</code></td>
<td><code>text</code>: Symbol up to 32 characters long.</td>
</tr>
<tr>
<td><code>,MODELNAME=</code></td>
<td><code>modelname</code>: Symbol up to eight characters long.</td>
</tr>
<tr>
<td><code>,MODELADDR=</code></td>
<td><code>modeladdr</code>: A-type address.</td>
</tr>
<tr>
<td><code>,FORMATNAME=</code></td>
<td><code>pgmname</code>: Symbol up to eight characters long.</td>
</tr>
<tr>
<td><code>,FORMATADDR=</code></td>
<td><code>pgmaddr</code>: A-type address.</td>
</tr>
<tr>
<td><code>,OFFSETASID=</code></td>
<td><code>ids</code>: One or more A-type addresses, separated by commas.</td>
</tr>
<tr>
<td>Syntax</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>OFFSETJOBNAME=(offsets)</td>
<td>Optional with EVENTDATA and not otherwise allowed. &lt;br&gt; offsets: One or more A-type addresses, separated by commas.</td>
</tr>
<tr>
<td>VIEWSUMMARY=rcode</td>
<td>Optional with EVENTDATA and not otherwise allowed. &lt;br&gt; rcode: A-type address.</td>
</tr>
<tr>
<td>VIEWFULL=fcode</td>
<td>Optional with EVENTDATA and not otherwise allowed. &lt;br&gt; fcode: A-type address.</td>
</tr>
<tr>
<td>COMPONENTDATA=cdata</td>
<td>Optional with EVENTDATA and not otherwise allowed. &lt;br&gt; cdata: A-type address.</td>
</tr>
<tr>
<td>EXCEPTION</td>
<td>Optional choice with EVENTDATA and not otherwise allowed.</td>
</tr>
<tr>
<td>NOEXCEPTION</td>
<td></td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

**MAP**

Specifies that a map of a format table is to be generated.

**TABLEDATA=tabname**

Specifies that the definition of an initialized format table is to be started. When you specify TABLEDATA, you also specify the name to be associated with the table and certain data that appears only once in the table.

**EVENTDATA=eventid**

Specifies the event identifier that is associated with a component trace event.

**TABLEEND**

Specifies the end of the definition of the format table.

**LOCBUFNAME=bufname**

Specifies the name of the locate buffer exit routine that is loaded by the IPCS CTRACE subcommand. IPCS calls this routine to locate a component's trace buffers in a dump.

**LOCBUFADDR=bufaddr**

Specifies the address of the locate buffer exit routine. IPCS calls this routine to locate a component's trace buffers in a dump.

**FILTERNAME=pgmname**

Specifies the name of the component filter exit routine that is loaded by the IPCS CTRACE subcommand. IPCS calls this routine to provide component-specific filtering for that component's trace entries. No component filter exit is supplied if you do not specify one.

**FILTERADDR=pgmaddr**

Specifies the address of the component filter exit routine. IPCS calls this routine to provide component-specific filtering for that component's trace entries. No component filter exit is supplied if you do not specify one.
,'ENTRYLENGTH=elength
   When elength is not zero, this parameter specifies the length of the fixed-length
component trace entries that the component maintains. When elength is zero, it
indicates that the component trace entries vary in length. A default of zero is
assumed.

,'MNEMONIC=mnemonic
   Specifies a mnemonic name for the type of event being described. This name is
the first information to be formatted on a line associated with an event entry of
this type. The name permits the reader of formatted component traces to
rapidly scan the output for patterns of events and events of particular interest.

,'DESCRIPTION=text
   Specifies descriptive, literal text to be associated with the type of trace entry
being described. When this type of trace entry is formatted, the text appears at
the end of the first line of the output. It helps the reader of the output to
understand the significance of an entry, without having to access separate
reference materials.

,'MODELNAME=modelname
   Specifies the name of the model that is to be used to format this trace entry.
   No model is used if MODELNAME or MODELADDR is not specified.

,'MODELADDR=modeladdr
   Specifies the address of the model to be used to format this trace entry. No
model is used if MODELADDR or MODELNAME is not specified.

,'FORMATNAME=pgmname
   Specifies the name of the formatter routine that formats this trace entry. No
formatter routine is called if FORMATNAME or FORMATADDR is not
specified.

,'FORMATADDR=pgmaddr
   Specifies the address of the formatter routine that formats this trace entry. No
formatter routine is called if FORMATADDR or FORMATNAME is not
specified.

,'OFFSETASID=(ids)
   If you want ASID filtering to be performed (as requested by an IPCS CTRACE
subcommand), use this parameter to specify the offsets to the ASID fields. The
ASID fields occur at various offsets in the trace entry. Specify up to 5 offsets.
   An offset value may not exceed decimal 65,535. If you do not specify
OFFSETASID, ASID filtering is not performed.

,'OFFSETJOBNAME=(offsets)
   If you want job name filtering to be performed (as requested by an IPCS
CTRACE subcommand), use this parameter to specify the offsets to the job
name fields. The job name fields occur at various offsets in the trace entry.
Specify up to 5 offsets. An offset value may not exceed decimal 65,535. If you
do not specify OFFSETJOBNAME, job name filtering is not performed.

,'VIEWSUMMARY=scode
   Specifies the halfword view that the model processor uses to format summary
fields from the trace entry. A default of X’8000’ for scode is used if you do not
specify this parameter.

,'VIEWFULL=fcode
   Specifies a halfword view (used by model processor) to format all fields from
the trace entry. A default of X’0200’ for fcode is used if you do not specify this
parameter.
ITTFMTB macro

,COMPONENTDATA=cdata
This parameter is reserved for use by the component. If this parameter is not specified, a default of zero is assumed for cdata indicating that no component data is associated with the trace entry.

,EXCEPTION
,NOEXCEPTION
EXCEPTION specifies that this trace entry records an exceptional event. When the IPCS CTRACE subcommand is invoked with the EXCEPTION filtering option, only trace entries with the EXCEPTION attribute are formatted.

NOEXCEPTION specifies that the trace entries being described record normal events. These entries will not be formatted when the IPCS CTRACE subcommand is invoked with the EXCEPTION of the filtering option. The default is NOEXCEPTION.

Return and reason codes
None.
Chapter 129. ITTWRITE — Write a full trace buffer to DASD or tape

Description

The ITTWRITE macro enables the component trace external writer to write a full trace buffer out to a trace data set on DASD or tape.

The ITTWRITE macro asynchronously captures a full trace buffer while the application continues processing and writing trace entries to another trace buffer.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Supervisor state or PSW key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB mode</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN=SASN or PASN¬=HASN¬=SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>64-bit.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled or disabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks may be held.</td>
</tr>
<tr>
<td>Control Parameters</td>
<td>Must be in the 64-bit primary address space.</td>
</tr>
</tbody>
</table>

Programming requirements

None.

Restrictions

If either the BUFFALET or the TBWCALET identifies the secondary or home address space, then both must identify the same address space (that is, both the trace buffer and the trace buffer writer control area must be in the same address space).

Register information

All registers are viewed as 64-bit values. After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the caller issued the macro. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>If GPR 15 contains 0 or 4, GPR 0 is used as a work register by the system; otherwise, GPR 0 contains a reason code.</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>
ITTWRITE macro

2-13 Unchanged
14 Used as a work register by the system
15 Return code

When control returns to the caller, the ARs contain:

Register          Contents

0-15 Unchanged

Performance implications
None.

Syntax

The standard form of the ITTWRITE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ITTWRITE.</td>
</tr>
<tr>
<td>ITTWRITE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ITTWRITE.</td>
</tr>
<tr>
<td>BUFFADDR=buffer_address</td>
<td>buffer_address: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td>,BUFFALET=buffer_alet</td>
<td>buffer_alet: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td>,BUFFALET=NOBUFFALET</td>
<td>Default: BUFFALET=NOBUFFALET</td>
</tr>
<tr>
<td>,BUFFLEN=buffer_length</td>
<td>buffer_length: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td>,TOKEN=token</td>
<td>token: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td>,TBWCADDR=tbwc_address</td>
<td>tbwc_address: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td>,TBWCADDR64=tbwc_address64</td>
<td>tbwc_address64: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td>,TBWCALET=tbwc_alet</td>
<td>tbwc_alet: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td>,TBWCALET=NOTBWCALET</td>
<td>Default: TBWCALET=NOTBWCALET</td>
</tr>
<tr>
<td>,SYNCH=YES</td>
<td>NO</td>
</tr>
<tr>
<td>,RC=return_code</td>
<td>return_code: RS-type address or register (2)-(12).</td>
</tr>
</tbody>
</table>
Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.RSNCODE=reason_code</td>
<td>reason_code: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td>.COM=comment</td>
<td>comment: A comment string.</td>
</tr>
<tr>
<td>.COM=NULL</td>
<td>Default: COM=NULL.</td>
</tr>
<tr>
<td>.MF=(S)</td>
<td>Default: MF=(S).</td>
</tr>
</tbody>
</table>

Parameters

The parameters are explained as follows:

BUFFADDR=buffer_address

Specifies a required parameter that points to the address of the buffer to be written externally.

BUFFALET=buffer_alet

BUFFALET=NOBUFFALET

Contains the PASN ALET that identifies the address/data space where the buffer resides. Use this optional parameter when the buffer to be written externally resides in either a data space or an address space that is different from the current primary address space. The default is BUFFALET=NOBUFFALET.

BUFFLEN=buffer_length

Specifies a required parameter that indicates the number of bytes in length of the buffer to be written externally. Though the buffer length is 64-bits, it is required to keep the buffer size within manageable limits. IBM suggest that the length be between 4KB and 512M. Component trace splits buffers that are too large to fit into a single block.

TOKEN=token

Specifies a required parameter that specifies the token passed to the start/stop exit routine when it was requested to start tracing externally.

TBWCADDR=tbwc_address

Specifies a required parameter that points to a word that points to the address of the storage obtained by the application for the trace buffer writer control area (TBWC) mapped by ITTTBWC. The TBWC provides communication between the application and component trace. See TBWC in z/OS MVS Data Areas in the z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkser/) for complete field names and lengths, offsets, and descriptions of the fields of the TBWC.

TBWCADDR64=tbwc_address64

Specifies a required parameter that points to a word that points to the address of the storage obtained by the application for the trace buffer writer control area (TBWC) mapped by ITTTBWC. The TBWC provides communication between the application and component trace. See TBWC in z/OS MVS Data Areas in the z/OS Internet library (http://www.ibm.com/systems/z/os/zos/bkser/) for complete field names and lengths, offsets, and descriptions of the fields of the TBWC.

TBWCALET=tbwc_alet
**ITTWRITE macro**

- **TBWCALET=NOTBWCALET**
  Contains the ALET that identifies the address/data space where the TBWC resides. Use this optional parameter when the TBWC resides in either a data space or an address space that is different from the current primary address space. The default is TBWCALET=NOTBWCALET.

- **SYNCH=YES | NO**
  YES causes CTRACE to copy the application's buffers before control is returned instead of scheduling an asynchronous SRB to copy the buffer. The ITTWRITE function executes synchronously. The SYNCH keyword is optional. NO causes the ITTWRITE function to execute asynchronously.

  **Note:** Because your application runs slower, IBM does not suggest that you use the SYNCH keyword on every ITTWRITE invocation. Use the SYNCH keyword in the start/stop routine any time that the trace buffers are to be freed. For example, when the trace is being turned off or the buffer size is changing, you can free trace buffer storage after issuing the ITTWRITE macro with the SYNCH keyword. The system copies the buffers to I/O buffers that CTRACE then can write to the external data set. The default is SYNCH=NO.

- **RC=return_code**
  Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15.

- **RSNCODE=reason_code**
  Specifies the location where the system is to store the reason code. If GPR 15 contains a return code other than 0 or 4, the reason code is also in GPR 0.

- **COM=comment**
  **COM=NULL**
  Comments the macro invocation. If the comment contains any lowercase characters, it must be enclosed in quotation marks.

- **MF=(S)**
  Specifies the standard form of the ITTWRITE macro.

**ABEND codes**

The following table identifies abend code and reason code combinations, and a description of what each means:

<table>
<thead>
<tr>
<th>Abend Code</th>
<th>Reason Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00D</td>
<td>00010100</td>
<td>For the ITTWRITE macro, the parameter list version number is not correct.</td>
</tr>
<tr>
<td>00D</td>
<td>00010200</td>
<td>The system found either nonzero values in the reserved fields or unused fields for the requested service in the ITTWRITE macro parameter list.</td>
</tr>
<tr>
<td>00D</td>
<td>00010300</td>
<td>The buffer length passed was 0 or less.</td>
</tr>
<tr>
<td>00D</td>
<td>00010400</td>
<td>The buffer length is unusually large and is not supported by CTRACE.</td>
</tr>
</tbody>
</table>

**Return and reason codes**

When control returns from ITTWRITE, GPR 15 (and return_code, if you coded RC) contains one of the following return codes. The third byte of GPR 0 (and reason_code, if you coded RSNCODE) might contain one of the following reason codes.
Note: An application should always check the return code from the ITTWRITE macro. A non-zero code indicates that some data might have been lost in the next record output.

Table 115. Return and Reason Codes for the ITTWRITE Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>None.</td>
<td>ITTWRITE was successful.</td>
</tr>
<tr>
<td>04</td>
<td>None.</td>
<td>ITTWRITE was unsuccessful. No data was captured because the trace is not connected to an active external writer.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx01xx</td>
<td>Storage required to perform the write operation could not be obtained.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx02xx</td>
<td>ITTWRITE was unable to schedule an SRB to process this request.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx03xx</td>
<td>The control information (TBWC) has already been reused by the application.</td>
</tr>
<tr>
<td>0C</td>
<td>xxxx01xx</td>
<td>The caller is holding locks.</td>
</tr>
<tr>
<td>0C</td>
<td>xxxx02xx</td>
<td>The input token was not valid.</td>
</tr>
<tr>
<td>0C</td>
<td>xxxx0300</td>
<td>The TBWC is not valid because the sequence number is the same as a previous write request.</td>
</tr>
</tbody>
</table>
| 0C                      | xxxx0301                | The TBWC is not valid for one of the following reasons:  
  • The TBWC is not in central storage and the ITTWRITE issuer is disabled.  
  • The BUFFALET is not the same as the TBWCALET. |

Example

Indicate to component trace that the buffer at address TRACEADR is ready to be written out. Pass the token (TCWTRTKN) that the application received from the start/stop routine. Component trace is to store the return and reason codes from the ITTWRITE macro in TCRCODE and TCRSNCODE.

```
ITTWRITE BUFFADDR=TRACEADR,BUFFLEN=TRACESIZ, X
TOKEN=TCWTRTKN,TBWCADDR=TBWCADR, X
RC=TCRCODE,RSNCODE=TCRSNCODE
```

<table>
<thead>
<tr>
<th>TBWCADR</th>
<th>DS A</th>
<th>TBWC address</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACEADR</td>
<td>DS A</td>
<td>Trace buffer address</td>
</tr>
<tr>
<td>TRACESIZ</td>
<td>DS F</td>
<td>Trace buffer size</td>
</tr>
<tr>
<td>TCWTRTKN</td>
<td>DS CL8</td>
<td>Trace writer token produced by CTRACE upon connection</td>
</tr>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCRCODE</td>
<td>DS F</td>
<td>Return code from CTRACE</td>
</tr>
<tr>
<td>TCRSNCODE</td>
<td>DS F</td>
<td>Reason code from CTRACE</td>
</tr>
</tbody>
</table>

**ITTWRITE - List form**

**Syntax**

The list form of the ITTWRITE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>*name: Symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ITTWRITE.</td>
</tr>
</tbody>
</table>
### ITTWRITE macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITTWRITE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ITTWRITE.</td>
</tr>
<tr>
<td>,MF=(L,cntl)</td>
<td><code>cntl</code>: Symbol.</td>
</tr>
<tr>
<td>,MF=(L,cntl,attr)</td>
<td><code>attr</code>: 1- to 60-character input string.</td>
</tr>
<tr>
<td>,MF=(L,cntl,0D)</td>
<td><strong>Default</strong>: 0D</td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

- `,MF=(L,cntl)`
- `,MF=(L,cntl,attr)`
- `,MF=(L,cntl,0D)`

Specifies the list form of the macro.

- `cntl` is the name of a storage area for the parameter list.
- `attr` is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code `attr`, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

### ITTWRITE - Execute form

Use the execute form of the ITTWRITE macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

#### Syntax

The execute form of the ITTWRITE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: Symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ITTWRITE.</td>
</tr>
<tr>
<td>ITTWRITE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ITTWRITE.</td>
</tr>
<tr>
<td>BUFFADDR=buffer_address</td>
<td><code>buffer_address</code>: RS-type address or register (2)-(12).</td>
</tr>
</tbody>
</table>
### ITTWRITE macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BUFFALET=buffer_alet</code></td>
<td><code>buffer_alet</code>: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td><code>BUFFALET=NOBUFFALET</code></td>
<td>Default: <code>BUFFALET=NOBUFFALET</code></td>
</tr>
<tr>
<td><code>BUFFLEN=buffer_length</code></td>
<td><code>buffer_length</code>: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td><code>TOKEN=token</code></td>
<td><code>token</code>: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td><code>TBWADDR=tbwc_address</code></td>
<td><code>tbwc_address</code>: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td><code>TBWADDR64=tbwc_address64</code></td>
<td><code>tbwc_address64</code>: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td><code>TBWCALET=tbwc_alet</code></td>
<td><code>tbwc_alet</code>: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td><code>TBWCALET=NOTBWCALET</code></td>
<td>Default: <code>TBWCALET=NOTBWCALET</code></td>
</tr>
<tr>
<td>`SYNCH=YES</td>
<td>NO`</td>
</tr>
<tr>
<td><code>RC=return_code</code></td>
<td><code>return_code</code>: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td><code>RSNCODE=reason_code</code></td>
<td><code>reason_code</code>: RS-type address or register (2)-(12).</td>
</tr>
<tr>
<td><code>COM=comment</code></td>
<td><code>comment</code>: A comment string.</td>
</tr>
<tr>
<td><code>COM=NULL</code></td>
<td>Default: <code>COM=NULL</code></td>
</tr>
<tr>
<td><code>MF=(E,cntl)</code></td>
<td><code>cntl</code>: RX-type address or register (2)-(12).</td>
</tr>
<tr>
<td><code>MF=(E,cntl,COMPLETE)</code></td>
<td>Default: <code>COMPLETE</code></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained under the standard form of the ITTWRITE macro with the following exception:

- `MF=(E,cntl)`
- `MF=(E,cntl,COMPLETE)`

  Specifies the execute form of the macro.

  `cntl` is the name of a storage area for the parameter list.

  COMPLETE specifies that the system is to check the macro parameter syntax and supply defaults on parameters that you do not use. COMPLETE is the default.
ITTWRITE macro
Chapter 130. ITZXFILT — Transaction trace filter exit

Description

The ITZXFILT macro is used to invoke the Transaction Trace filter exit.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state. PSW key 8 - 15</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
</tbody>
</table>

If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.

Interrupt status: Enabled for I/O and external interrupts

Locks: Unlocked or locked

Control parameters: Control parameters must be in the primary address space.

Programming requirements

None.

Restrictions

1. Addressability to CVT and IHAECVT must be established prior to using this macro.
2. FRRs are allowed.
3. The version of the IWMCLSFY parameter list must be 4 or higher.

Input register information

Before issuing the ITZXFILT macro, the caller must insure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of a 72-byte standard save area in the primary address space</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Unpredictable (Used as a work register by the system)</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Unpredictable (Used as a work register by the system)</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>
When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Unpredictable (Used as a work register by the system)</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Unpredictable (Used as a work register by the system)</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a macro. If the macro changes the contents of registers on which the caller depends, the caller must save them before issuing the macro and restore them after the macro returns control.

Performance implications
None.

Syntax
The ITZXFILT macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede ITZXFILT.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ITZXFILT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow ITZXFILT.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>FILTPARM=filtparm</td>
<td>filtparm: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>,WKAREA=wkarea</td>
<td>wkarea: RS-type address or address in register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or address in register (2) - (12).</td>
</tr>
</tbody>
</table>

Parameters
The parameters are explained as follows:

name
This is an optional symbol, starting in column 1, that is the name on the ITZXFILT macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

FILTPARM=filtparm
This is a required input parameter of the IWMCLSFY parameter list to be passed to the transaction trace filter exit.
ITZXFILT macro

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,WKAREA=wkarea
This is a required input parameter of a 256-byte work area to be used by the transaction trace filter exit routine.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,RETCODE=retcode
This is an optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

ABEND codes
None.

Return and reason codes
When the ITZXFILT macro returns control to your program, GPR 15 (and retcode, when you code RETCODE) contains a return code.

The following table identifies the hexadecimal return and reason codes.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Meaning: The transaction trace token was created. Action: None.</td>
</tr>
<tr>
<td>4</td>
<td>Meaning: The transaction trace token was not created. Action: If the address of the 32-character transaction trace token field was not provided, provide it in the parameter list. Otherwise, the token was not created because of no match between the input parameter attributes and the transaction trace filter sets attributes.</td>
</tr>
</tbody>
</table>

Example

IWMCLSFY MF=(L,TT_PARM),PLISTVER=MAX
USING TT_PARM,R5
LA R1,TRANNAME
ST R1,TT_PARM_TRANTOKN_ADDR
LA R1,TT_PARM_XTRACETOKEN_ADDR
ST R1,TT_PARM_XTRACETOKEN_ADDR
ITZXFILT FILTPARM=TT_PARM,WKAREA=TT_WORK
SPACE 1
LTR R15,R15
BC NZERO,NOTOKEN
.
.
NOTOKEN DS 0H
.
.
TT WORK DS CL256
TRANNAME DC CL8'MYWORKNM'
TRANTOKN DC CL32''
ITZFILT macro
Chapter 131. IXGBRWSE — Browse/read a log stream

Description

Use the IXGBRWSE macro to read and browse a log stream for log block information. Using IXGBRWSE, a program can read consecutive log blocks in a log stream or search for and read a specific log block in a log stream. IXGBRWSE returns the specified log block in the calling program’s output buffer.

The requests for IXGBRWSE are:

- **REQUEST=START**, which starts a browse session. A browse session is identified by a browse token which is created by the browse start request. The browse session remains active until it is ended as a result of a REQUEST=END request or the log stream has been disconnected. See “REQUEST=START option of IXGBRWSE” on page 1288 for the syntax of this request.

- **REQUEST=READCURSOR**, which reads the next consecutive log block (or blocks) in the log stream. Use this request multiple times or use the MULTIBLOCK keyword to read consecutive blocks in a log stream. See “REQUEST=READCURSOR option of IXGBRWSE” on page 1294 for the syntax of this request.

- **REQUEST=READBLOCK**, which reads a selected log block in a log stream. See “REQUEST=READBLOCK option of IXGBRWSE” on page 1301 for the syntax of this request.

- **REQUEST=RESET**, which resets the browse cursor to either the beginning or the end of the log stream. See “REQUEST=RESET option of IXGBRWSE” on page 1308 for the syntax of this request.

- **REQUEST=END**, which ends a browse session. See “REQUEST=END option of IXGBRWSE” on page 1313 for the syntax of this request.

For information about using the system logger services and the IXGBRWSE request, see [z/OS MVS Programming: Assembler Services Guide](https://www.ibm.com/support/docview.wss?uid=swg27028566), which also includes information about related macros IXGCONN, IXGINVNT, IXGWRITE, IXGDELETE, and IXGQUERY.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem or Supervisor state with any PSW key. The caller must be in supervisor state with any system (0-7) PSW key to either invoke this service in SRB mode or to use the MODE=SYNCEXIT keyword.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, HASN or SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held.</td>
</tr>
</tbody>
</table>
### Environmental factor

**Control parameters:**

- All control parameters must be in the primary address space with the following exceptions:
  - The ECB should be addressable from the home address space.
  - Any parameter area that is explicitly ALET-qualified as allowed by the input parameter (for example, the area referenced by the BUFFER parameter when the BUFFALET parameter is specified) must be in an address or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

All storage areas specified must be in the same storage key as the caller with the following exception:

- Any parameter area is explicitly storage key qualified as allowed by the input parameters (example: the area referenced by the BUFFER parameter when the BUFFKEY parameter is also specified).

### Programming requirements

- The current primary address space must be the same primary address space used at the time your program issued the IXGCONN request.
- The calling program must be connected to the log stream through the IXGCONN service with either read or write authority.
- The parameter list for this service must be addressable in the caller's primary address space.
- Include the IXGCON mapping macro in your program. This macro provides a list of equate symbols for the system logger services.
- Include macro IXGANSAA in your program. This macro maps the format of the answer area output returned for each system logger service in the ANSAREA parameter.
- For a READCURSOR browse request with the MULTIBLOCK=YES option, include the IXGBRMLT mapping macro in your program. This macro provides a mapping of the area returned by the system logger for each block that is returned in the caller's buffer. Additionally, the area pointed to by the BUFFER or BUFFER64 parameter must be on a word boundary for multiple log block READCURSOR requests.
- Although the data pointed to by the BUFFER64 keyword may be above the bar (2-gigabyte), the length of the name or address of the input field specified in the BUFFLEN keyword is still limited to 4 bytes.
- When coding the MODE=SYNCECB and ECB parameters, you must ensure that:
  - The virtual storage area specified for the ECB resides on a fullword boundary.
  - You initialize the ECB field to zero.
  - The ECB resides in either common or home address space storage at the time the IXGBRWSE request is issued.
  - The storage used for output parameters, such as ANSAREA, BROWSETOKEN, BUFFER, BUFFER64, ANSAREA, BLKSIZE, TIMESTAMP, and RETBLOCKID, are accessible by both the IXGBRWSE invoker and the ECB waiter.
- When coding the MODE=SYNCEXIT parameter, you must ensure that the storage used for output parameters, such as ANSAREA, BROWSETOKEN,
BUFFER, BUFFER64, ANSAREA, BLKSIZE, TIMESTAMP, and RETBLOCKID, are accessible by both the IXGBRWSE invoker and the completion exit routine.

Restrictions

There is more than one version of this macro available. The parameters you can use depend on the version you specify on the PLISTVER parameter. See the description of the PLISTVER parameter for more information.

You can call any of the system logger services in either AMODE 31 or 64, but the parameter list and all other data addresses, with the exception of BUFFER64 must reside in 31-bit storage.

Input register information

Before issuing the IXGBRWSE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if register 15 contains a non-zero return code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

When control returns to a caller running in AMODE 64, the 64–bit registers contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as a work register by the system, if the caller specified BUFFER64. Otherwise, unchanged.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>
**IXGBRWSE macro**

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.

**REQUEST=START option of IXGBRWSE**

The IXGBRWSE macro with the REQUEST=START parameter starts a browse session and sets the starting position of the browse cursor.

**Syntax for REQUEST=START**

The IXGBRWSE REQUEST=START macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IXGBRWSE.</td>
</tr>
<tr>
<td>IXGBRWSE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IXGBRWSE.</td>
</tr>
<tr>
<td>REQUEST=START</td>
<td></td>
</tr>
<tr>
<td>,STREAMTOKEN=streamtoken</td>
<td>streamtoken: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BROWSETOKEN=browsetoken</td>
<td>browsetoken: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ANSAREA=ansarea</td>
<td>ansarea: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ANSLLEN=anslen</td>
<td>anslen: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,OLDEST</td>
<td>Default: OLDEST</td>
</tr>
<tr>
<td>,YOUNGEST</td>
<td></td>
</tr>
<tr>
<td>,STARTBLOCKID=startblockid</td>
<td>startblockid: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,SEARCH=search</td>
<td>search: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>GMT=YES</td>
<td></td>
</tr>
<tr>
<td>GMT=NO</td>
<td></td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIEW=ACTIVE</td>
<td>Default: VIEW=ACTIVE</td>
</tr>
<tr>
<td>VIEW=ALL</td>
<td></td>
</tr>
<tr>
<td>VIEW=NO_VIEW</td>
<td></td>
</tr>
<tr>
<td>MODE=SYNC</td>
<td>Default: MODE=SYNC</td>
</tr>
<tr>
<td>MODE=SYNCECB</td>
<td></td>
</tr>
<tr>
<td>MODE=SYNCEXIT</td>
<td></td>
</tr>
<tr>
<td>REQDATA=reqdata</td>
<td>reqdata: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>ECB=ecb</td>
<td>ecb: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>DIAG=NO_DIAG</td>
<td>Default: DIAG=NO_DIAG</td>
</tr>
<tr>
<td>DIAG=NO</td>
<td></td>
</tr>
<tr>
<td>DIAG=YES</td>
<td></td>
</tr>
<tr>
<td>PLISTVER=IMPLIED_VERSION</td>
<td></td>
</tr>
<tr>
<td>PLISTVER=MAX</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>PLISTVER=plistver</td>
<td></td>
</tr>
<tr>
<td>RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>MF=(L,list addr)</td>
<td></td>
</tr>
<tr>
<td>MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
<tr>
<td>MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
<tr>
<td>MF=(M,list addr)</td>
<td></td>
</tr>
<tr>
<td>MF=(M,list addr,COMPLETE)</td>
<td></td>
</tr>
<tr>
<td>MF=(M,list addr,NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters for REQUEST=START

The parameters are explained as follows:

**REQUEST=START**

Requests that a browse session be started.
IXGBRWSE macro

\,STREAMTOKEN=streamtoken

Specifies the name or address (using a register) of a required 16-byte input field containing the token for the log stream that you want to browse and read. The stream token is returned by the IXGCONN service at connection to the log stream.

\,BROWSETOKEN=browsetoken

Specifies the name or address (using a register) of a required 4-byte output area where a token uniquely identifying the browse session is returned by the IXGBRWSE REQUEST=START request. This browse token is then used as an input to subsequent IXGBRWSE requests to identify the browse session.

\,ANSAREA=ansarea

Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSA macro.

\,ANSLEN=anslen

Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSA macro.

\,OLDEST
\,YOUNGEST
\,STARTBLOCKID=startblockid
\,SEARCH=search

Specifies where the cursor should be set for the start of the browse session.

- **OLDEST**: Specifies that the block cursor be positioned at the oldest log block in the log stream.
  
  When VIEW=ACTIVE is specified for this browse session, the cursor is positioned at the oldest active log block in the log stream. If there is no active data in the log stream, the request will fail.
  
  When VIEW=ALL is specified, the cursor is positioned at the oldest log block in the log stream of the active and inactive data. If there is neither active nor inactive data in the log stream, the request will fail.

- **YOUNGEST**: Specifies that the block cursor be positioned at the youngest log block in the log stream.
  
  When VIEW=ACTIVE is specified for this browse session, the cursor is positioned at the youngest active log block in the log stream.
  
  When VIEW=ALL is specified, the cursor is positioned at the youngest log block in the log stream, even if the youngest block is eligible for deletion.

- **STARTBLOCKID=startblockid**: Specifies the name (or register) of a 8-byte input field containing the block identifier for the log block you want to use as the starting cursor position.
  
  When VIEW=ALL is specified, you must specify a starting block that is active.

- **SEARCH=search**: Specifies the name (or register) of a 64-bit input field containing the time stamp you want to use in searching for a particular log block as the starting cursor position for this browse session. For information on how the SEARCH keyword works, see z/OS MVS Programming: Assembler Services Guide.
The time stamp must be Coordinated universal time (UTC) or local time, in time of day (TOD) clock format. The GMT parameter is required with the SEARCH parameter.

- **GMT=YES**
  - The time stamp specified on the SEARCH parameter is in UTC format.
- **GMT=NO**
  - The time stamp specified on the SEARCH parameter is local time.

**VIEW=ACTIVE**
**VIEW=ALL**
**VIEW=NO_VIEW**

Specifies whether requests issued during this browse session return active data only, or both active and inactive data. Active data is data that has not been marked for deletion via the IXGDELETE service. Inactive data is data that has been deleted via IXGDELETE but has not been physically deleted from the log stream because of the retention period specified in the log stream definition in the LOGR couple data set.

- **VIEW=ACTIVE**, which is the default, specifies that in this browse session, system logger will only return active data from the log stream.
- **VIEW=ALL** specifies that in this browse session, system logger will return both active and inactive data.
  - When VIEW=ALL is specified and a log block is returned, system logger sets a flag in the answer area, AnsaBlkFromlnactive, indicating whether the block was active or eligible for deletion.
- **VIEW=NO_VIEW** specifies that the default VIEW value will be used for the browse session.

The system where IXGBRWSE is issued must be IPLed for the VIEW parameter to be recognized.

- **MODE=SYNC**
- **MODE=SYNCECB**
- **MODE=SYNCEXIT**

Specifies that the request should be processed in one of the following ways:

- **MODE=SYNC**: Specifies that the request process synchronously. Control is not returned to the caller until request processing is complete. If necessary, the calling program will be suspended until the request completes.
- **MODE=SYNCECB**: Specifies that the request process synchronously if possible. If the request processes asynchronously, control returns to the caller before the request completes and the event control block (ECB) specified on the ECB parameter is posted when the request completes. The ECB parameter is required with MODE=SYNCECB.
- **MODE=SYNCEXIT**: Specifies that the request process synchronously, if possible. If the request cannot be processed synchronously, your complete exit (specified on the COMPLETEEXIT parameter on the IXGCONN request) gets control when this request completes. Control returns to the caller with a return and reason code indicating that the request is not complete. The system passes the data specified on the REQDATA parameter, if specified, to the complete exit.
IXGBRWSE macro

When a MODE=SYNCEXIT request processes asynchronously, system logger
maintains latent binds to the storage location specified by the answer area
(ANSAREA) fields, and, if specified, to RETBLOCKID and TIMESTAMP.
To use this parameter, the system where the application is running must be
IPLed. The application must run in supervisor state, key 0-7 to use this
parameter.

,ECB=ecb
Specifies the name or address (using a register) of a 4-byte input field
containing an event control block (ECB) to be posted when the request
completes.
Before coding ECB, you must ensure that:
• You initialize the ECB to zero.
• The ECB must reside in either common storage or the home address space at
  the time the IXGBRWSE request is issued.
• The virtual storage area specified for the ECB must reside on a fullword
  boundary.

,DIAG=NO_DIAG
,DIAG=NO
,DIAG=YES
Specifies whether or not the DIAG option on the IXGCONN for this logstream
will be in effect for this browse session. Refer to the DIAG keyword on the
IXGINVNT, IXGCONN, and IXGDELET macro services.
If you specify DIAG=NO_DIAG, which is the default, then the DIAG option
on the IXGCONN for this logstream will be in effect for this browse session.
If you specify DIAG=NO, then Logger will not take additional diagnostic action
as defined in the logstream definition DIAG parameter.
If you specify DIAG=YES, then Logger will take additional diagnostic action as
defined on the logstream definition DIAG parameter providing the IXGCONN
connect DIAG specification allows it.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=plistver
An optional input parameter that specifies the version of the macro. PLISTVER
determines which parameter list the system generates.
The values are:
• IMPLIED_VERSION, which is the lowest version that allows all parameters
  specified on the request to be processed. If you omit the PLISTVER
  parameter, IMPLIED_VERSION is the default. Note that on the list form, the
default will cause the smallest parameter list to be created.
• MAX, if you want the parameter list to be the largest size currently possible.
  This size might grow from release to release and affect the amount of
  storage that your program needs.
  If you can tolerate the size change, IBM recommends that you always
  specify PLISTVER=MAX on the list form of the macro. Specifying MAX
  ensures that the list-form parameter list is always long enough to hold all
  the parameters you might specify on the execute form when both forms are
  assembled using the same level of the system. In this way, MAX ensures that
  the parameter list does not overwrite nearby storage.
• 0, supports all parameters except those specifically referenced in higher
  versions.
• 1, supports both the following parameters and parameters from version 0:
  – DIAG
  – REQDATA
• 2, supports both the following parameters and parameters from version 0 and 1:
  – MAXNUMLOGBLOCKS
  – MULTIBLOCK
  – RETBLOCKINFO

To code: Specify in this input parameter one of the following:
• IMPLIED_VERSION
• MAX
• A decimal value of 0, 1 or 2

,RETCODE=retcode
  Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=rsncode
  Specifies a name or address (using a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

,MF=S
,MF=(L,list addr)
,MF=(L,list addr,attr)
,MF=(L,list addr,OD)
,MF=(E,list addr)
,MF=(E,list addr,COMPLETE)
,MF=(E,list addr,NOCHECK)
,MF=(M,list addr)
,MF=(M,list addr,COMPLETE)
,MF=(M,list addr,NOCHECK)

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

You should use the modify and execute forms in the following order:
IXGBRWSE macro

- Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,list_addr,NOCHECK), to execute the macro.

,,list addr
The name of a storage area to contain the parameters.

,,attr
An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

REQUEST=READCURSOR option of IXGBRWSE
The IXGBRWSE macro with the REQUEST=READCURSOR option allows a program to read the next consecutive log block in a log stream. Subsequent READCURSOR requests will start reading at the next consecutive block. Use this request multiple times or use the MULTIBLOCK keyword to read a series of consecutive log blocks. The direction of the browse is controlled by the program and can be changed dynamically.

READCURSOR requests are limited to reading log blocks within the range of data defined by the browse session’s view. The view is controlled by the VIEW keyword on either the browse START request or the browse RESET request.

Note: REQUEST=READCURSOR reads the next consecutive log block in the log stream, but the blocks may not be in exact local time sequence. This can happen, for example, because of daylight savings time, one or more records with the same local time stamp, or multiple applications writing to the same log stream.

Syntax for REQUEST=READCURSOR
The IXGBRWSE REQUEST=READCURSOR macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IXGBRWSE.</td>
</tr>
<tr>
<td>IXGBRWSE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IXGBRWSE.</td>
</tr>
</tbody>
</table>
### IXGBRWSE macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST=READCURSOR</td>
<td></td>
</tr>
<tr>
<td>,STREAMTOKEN=streamtoken</td>
<td><code>streamtoken</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BROWSETOKEN=browsetoken</td>
<td><code>browsetoken</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BUFFER=buffer</td>
<td><code>buffer</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BUFFER64=buffer64</td>
<td><code>buffer64</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BUFFLEN=buffers</td>
<td><code>buffers</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,DIRECTION=OLDTOYOUNG</td>
<td></td>
</tr>
<tr>
<td>,DIRECTION=YOUNGTOOLD</td>
<td></td>
</tr>
<tr>
<td>,ANSAREA=ansarea</td>
<td><code>ansarea</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ANSLEN=anslen</td>
<td><code>anslen</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BUFFKEY=buffkey</td>
<td><code>buffkey</code>: RS-type address or register (2) - (12). Default: PSW key of the caller</td>
</tr>
<tr>
<td>,BUFFALET=buffalet</td>
<td><code>buffalet</code>: RS-type address or register (2) - (12). Default: BUFFALET=0</td>
</tr>
<tr>
<td>,BLKSIZE=blksize</td>
<td><code>blksize</code>: RS-type address or register (2) - (12). Default: BLKSIZE=0</td>
</tr>
<tr>
<td>,MULTIBLOCK=YES</td>
<td></td>
</tr>
<tr>
<td>,MULTIBLOCK=NO</td>
<td>Default: MULTIBLOCK=NO</td>
</tr>
<tr>
<td>,RETBLOCKID=retblockid</td>
<td><code>retblockid</code>: RS-type address or register (2) - (12). Default: NO_BLKID Note: RETBLOCKID is valid with MULTIBLOCK=NO only.</td>
</tr>
<tr>
<td>,TIMESTAMP=timestamp</td>
<td><code>timestamp</code>: RS-type address or register (2) - (12). Default: NO_TIMESTAMP Note: TIMESTAPM is valid with MULTIBLOCK=NO only.</td>
</tr>
<tr>
<td>,RETBLOCKINFO=YES</td>
<td></td>
</tr>
<tr>
<td>,RETBLOCKINFO=NO</td>
<td>Default: NO Note: RETBLOCKINFO is valid with MULTIBLOCK=YES only.</td>
</tr>
</tbody>
</table>
IXGBRWSE macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>
| \(\text{MAXNUMLOGBLOCKS}=\text{maxnumlogblocks}\) | \(\text{maxnumlogblocks}\): RS-type address or register (2) - (12).
| \(\text{MAXNUMLOGBLOCKS}=0\) | Note: \(\text{MAXNUMLOGBLOCKS}\) is valid with \(\text{MULTIBLOCK=YES}\) only. |
| \(\text{MODE=SYNC}\) | Default: \(\text{MODE=SYNC}\) |
| \(\text{MODE=SYNCECB}\) | |
| \(\text{MODE=SYNCEXIT}\) | |
| \(\text{ECB}=\text{ecb}\) | \(\text{ecb}\): RS-type address or register (2) - (12). |
| \(\text{ECB}=\text{ecb}\) | |
| \(\text{PLISTVER=IMPLIED\_VERSION}\) | Default: \(\text{IMPLIED\_VERSION}\) |
| \(\text{PLISTVER=MAX}\) | |
| \(\text{PLISTVER}=\text{plistver}\) | |
| \(\text{RETCODE}=\text{retcode}\) | \(\text{retcode}\): RS-type address or register (2) - (12). |
| \(\text{RSNCODE}=\text{rsncode}\) | \(\text{rsncode}\): RS-type address or register (2) - (12). |
| \(\text{MF}=\text{S}\) | Default: \(\text{MF=\ S}\) |
| \(\text{MF}=(\text{L, list addr})\) | |
| \(\text{MF}=(\text{L, list addr, attr})\) | |
| \(\text{MF}=(\text{L, list addr, 0D})\) | |
| \(\text{MF}=(\text{E, list addr})\) | |
| \(\text{MF}=(\text{E, list addr, COMPLETE})\) | |
| \(\text{MF}=(\text{E, list addr, NOCHECK})\) | |
| \(\text{MF}=(\text{M, list addr})\) | |
| \(\text{MF}=(\text{M, list addr, COMPLETE})\) | |
| \(\text{MF}=(\text{M, list addr, NOCHECK})\) | |

Parameters for REQUEST=READCURSOR

The parameters are explained as follows:

REQUEST=READCURSOR

Requests that a program read the next consecutive log block in the log stream, in the direction specified on the DIRECTION parameter.

STREAMTOKEN=streamtoken

Specifies the name or address (using a register) of a required 16-byte input
field containing the token for the log stream that you want to browse and read. The stream token is returned by the IXGCONN service at connection to the log stream.

\texttt{,BROWSETOKEN=browsetoken}

Specifies the name or address (using a register) of a required 4-byte input field containing the identifier for the browse session which was returned on the IXGBRWSE REQUEST=START request.

\texttt{,BUFFER=buffer, BUFFER64=buffer64}

Specifies the name or address (using a register) of a required output field that contains the buffer into which the log block is read.

- \texttt{BUFFER=buffer} specifies that the location of the buffer is in 31-bit storage.
- \texttt{BUFFER64=buffer64} specifies that the location of the buffer is in 64-bit storage.

the BUFFER and BUFFER64 parameters are mutually exclusive.

\texttt{,BUFFLEN=bufflen}

Specifies the name or address (using a register) of a required 4-byte input field that contains the length of the buffer specified on the BUFFER or BUFFER64 parameter.

IXGBRWSE will return the length of the block in the BLKSIZE parameter, if specified. If you specify MULTIBLOCK=NO, you can issue IXGBRWSE with BLKSIZE specified to obtain the length of the block and then re-issue IXGBRWSE using the returned BLKSIZE value in the BUFFLEN parameter.

\texttt{,DIRECTION=OLDTOYOUNG, DIRECTION=YOUNGTOOLD}

Specifies the direction that you want the cursor to move to read the next consecutive log block. Specify OLDTOYOUNG to get the next youngest block or YOUNGTOOLD to get the next oldest block.

\texttt{,ANSAREA=ansarea}

Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

\texttt{,ANSLEN=anslen}

Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA\_PREFERRED\_SIZE field of the IXGANSAA macro.

\texttt{,BUFFKEY=buffkey}

Specifies the name (or address in a register) of a 4-byte input field specifying the storage key for the buffer specified on the BUFFER or BUFFER64 parameter.

If the caller is running in problem state, the caller’s PSW key and the key specified in the BUFFKEY parameter must match.

If the caller is running in supervisor state, specify any syntactically valid (0 through 15) key on the BUFFKEY parameter.

If you omit the BUFFKEY parameter, the default used is the PSW key of the caller.
BUFFALET=bufalet
Specifies the name (or address in a register) of a 4-byte input field specifying the access list entry table (ALET) to be used to access the buffer specified on the BUFFER or BUFFER64 keyword. If the buffer is ALET-qualified, the ALET must index a valid entry on the task's dispatchable unit access list (DUAL) or specify a SCOPE=COMMON data space. An ALET that indexes the system logger PASN-AL list will not work.

The default is 0, which means that the buffer is in the calling program's primary address space.

BLKSIZE=blksize
Specifies the name or address (using a register) of a 4-byte output field where the space used or needed in the BUFFER or BUFFER64 area is returned. When MULTIBLOCK=NO is specified and there is enough space in the buffer to return the requested log block data, the actual size of the log block is returned. When MULTIBLOCK=YES is specified and there is enough space in the buffer to return the requested log blocks, the amount of space used in the BUFFER or BUFFER64 area is returned. If the BUFFLEN value is not large enough to allow any log block data to be returned, then the BLKSIZE value will indicate the minimum amount of space necessary to return the next log block.

MULTIBLOCK=YES
MULTIBLOCK=NO
Specifies whether one or more than one log stream log block will be returned by the read cursor request.

- MULTIBLOCK=NO indicates that only one log stream log block is to be returned.
- MULTIBLOCK=YES indicates that the system logger will retrieve as many log blocks as meet the browse parameter criteria and fit into the caller's buffer.

RETBLOCKID=retblockid
Specifies the name or address (using a register) of an 8-byte output field where the identifier or the requested log block is returned.

TIMESTAMP=timestamp
Specifies the name or address (using a register) of a 16-byte output field where the Coordinated universal time stamp and the local time stamp associated with the requested log block are returned. The UTC time stamp is first, then the local time stamp. Both time stamps are in TOD-clock format.

RETBLOCKINFO=YES
RETBLOCKINFO=NO
Specifies whether or not system logger should return the log blocksize, blockid, timestamps and other identification information in the caller's buffer as part of the output. Specify RETBLOCKINFO=YES to receive each log block's identification information. Specify RETBLOCKINFO=NO to only receive the information necessary to navigate the caller's buffer.

If you omit the RETBLOCKINFO parameter, RETBLOCKINFO=NO is the default.

MAXNUMLOGBLOCKS=xmaxnumlogblocks
Specifies the name (or address in a register) of an optional fullword input that indicates the maximum number of log blocks to be returned in the buffer. When a non-zero value is specified, system logger will not return more than this requested number of log blocks, even if there are more log blocks that meet the other browse parameter criteria.
• If enough room is provided in the BUFFLEN value and there are sufficient log blocks that meet the browse criteria, system logger will return the requested maximum number of log blocks.
• If enough room is not provided in the BUFFLEN value, system logger will return as many log blocks as fit into the caller’s buffer.
• If there are fewer log blocks remaining than the requested maximum number, system logger will return as many of the remaining log blocks as fit into the caller’s buffer.

If you omit the MAXNUMLOGBLOCKS, the default is 0.

,MODE=SYNC
,MODE=SYNCECB
,MODE=SYNCEXIT

Specifies that the request should be processed in one of the following ways:
• MODE=SYNC: Specifies that the request process synchronously. Control is not returned to the caller until request processing is complete. If necessary, the calling program will be suspended until the request completes.
• MODE=SYNCECB: Specifies that the request process synchronously if possible. If the request processes asynchronously, control returns to the caller before the request completes and the event control block (ECB) specified on the ECB parameter is posted when the request completes. The ECB parameter is required with MODE=SYNCECB.
• MODE=SYNCEXIT: Specifies that the request process synchronously, if possible. If the request cannot be processed synchronously, your complete exit (specified on the COMPLETEEXIT parameter on the IXGCONN request) gets control when this request completes. Control returns to the caller with a return and reason code indicating that the request is not complete. The system passes the data specified on the REQDATA parameter, if specified, to the complete exit.

When a MODE=SYNCEXIT request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to RETBLOCKID and TIMESTAMP.

To use this parameter, the system where the application is running must be IPLed. The application must run in supervisor state, key 0-7 to use this parameter.

ECB=ecb

Specifies the name or address (using a register) of a 4-byte input field that contains an event control block (ECB) to be posted when the request completes.

Before coding ECB, you must ensure that:
• You initialize the ECB to zero.
• The ECB must reside in either common storage or the home address space at the time the IXGBRWSE request is issued.
• The virtual storage area specified for the ECB must reside on a fullword boundary.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=plistver

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:
**IXGBRWSE macro**

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- 0, supports all parameters except those specifically referenced in higher versions.

- 1, supports both the following parameters and parameters from version 0:
  - DIAG
  - REQDATA

- 2, supports both the following parameters and parameters from version 0 and 1:
  - MAXNUMLOGBCKS
  - MULTIBLOCK
  - RETBLCINFO

To code: Specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1 or 2

\[ \text{RETCODE=retcode} \]

Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

\[ \text{RSNCODE=rsncode} \]

Specifies a name or address (using a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

\[ \text{MF=S} \]

\[ \text{MF=(L,list addr)} \]

\[ \text{MF=(L,list addr,attr)} \]

\[ \text{MF=(L,list addr,0D)} \]

\[ \text{MF=(E,list addr)} \]

\[ \text{MF=(E,list addr,COMPLETE)} \]

\[ \text{MF=(E,list addr,NOCHECK)} \]

\[ \text{MF=(M,list addr)} \]

\[ \text{MF=(M,list addr,COMPLETE)} \]

\[ \text{MF=(M,list addr,NOCHECK)} \]

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The
list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:
- Use MF=(M, list_addr, COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M, list_addr, NOCHECK), specifying the parameters you want to change.
- Use MF=(E, list_addr, NOCHECK), to execute the macro.

list addr
The name of a storage area to contain the parameters.

attr
An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

REQUEST=READBLOCK option of IXGBRWSE
The IXGBRWSE macro with the REQUEST=READBLOCK parameter allows a program to search for and read a specific log block from the log stream. The target can be defined either by the log block identifier or by a time stamp.

Syntax for REQUEST=READBLOCK
The IXGBRWSE REQUEST=READBLOCK macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IXGBRWSE.</td>
</tr>
</tbody>
</table>
### IXGBRWSE macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXGBRWSE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IXGBRWSE.</td>
</tr>
<tr>
<td>REQUEST=READBLOCK</td>
<td></td>
</tr>
<tr>
<td>,STREAMTOKEN=stramtoken</td>
<td><em>streamtoken</em>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BROWSETOKEN=browsetoken</td>
<td><em>browsetoken</em>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BLOCKID=blockid</td>
<td><em>blockid</em>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,SEARCH=search</td>
<td><em>search</em>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BUFFER=buffer</td>
<td><em>buffer</em>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BUFFER64=buffer64</td>
<td><em>buffer64</em>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BUFFLEN=bufflen</td>
<td><em>bufflen</em>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ANSAREA=ansarea</td>
<td><em>ansarea</em>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ANSLEN=anslen</td>
<td><em>anslen</em>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>GMT=YES</td>
<td></td>
</tr>
<tr>
<td>GMT=NO</td>
<td></td>
</tr>
<tr>
<td>,BUFFKEY=buffkey</td>
<td><em>buffkey</em>: RS-type address or register (2) - (12). <strong>Default</strong>: PSW key of the caller</td>
</tr>
<tr>
<td>,BUFFALET=buffalet</td>
<td><em>buffalet</em>: RS-type address or register (2) - (12). <strong>Default</strong>: BUFFALET=0</td>
</tr>
<tr>
<td>,BLKSIZE=blksize</td>
<td><em>blksize</em>: RS-type address or register (2) - (12). <strong>Default</strong>: BLKSIZE=0</td>
</tr>
<tr>
<td>,RETBLOCKID=rethblockid</td>
<td><em>rethblockid</em>: RS-type address or register (2) - (12). <strong>Default</strong>: NO_BLKID</td>
</tr>
<tr>
<td>,TIMESTAMP=timestamp</td>
<td><em>timestamp</em>: RS-type address or register (2) - (12). <strong>Default</strong>: NO_TIMESTAMP</td>
</tr>
<tr>
<td>MODE=SYNC</td>
<td><strong>Default</strong>: MODE=SYNC</td>
</tr>
</tbody>
</table>
Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODE=SYNCECB</td>
<td></td>
</tr>
<tr>
<td>MODE=SYNCEXIT</td>
<td></td>
</tr>
<tr>
<td>,ECB=ecb</td>
<td>ecb: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=plistver</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
<tr>
<td>,MF=(M,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(M,list addr,COMPLETE)</td>
<td></td>
</tr>
<tr>
<td>,MF=(M,list addr,NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>

Parameters for REQUEST=READBLOCK

The parameters are explained as follows:

**REQUEST=READBLOCK**

Requests that a program read a specific block from the log stream. The target can be defined either by the log block identifier or by a time stamp.

**STREAMTOKEN=streamtoken**

Specifies the name or address (using a register) of a required 16-byte input field containing the token for the log stream that you want to search. The stream token is returned by the IXGCONN service at connection to the log stream.

**BROWSETOKEN=browsetoken**

Specifies the name or address (using a register) of a required 4-byte input field containing the identifier for the browse session which was returned from the IXGBRWSE REQUEST=START request.

**BLOCKID=blockid**

Specifies the name or address (using a register) of an 8-byte input field that contains the block identifier of the log block you wish to read. The block identifier was returned from the IXGWRITE request.
zhong

IXGBRWSE macro

\, SEARCH=search

Specifies the name or address (using a register) of a 64-bit input field containing the time stamp for the log block you wish to search for and read. The time stamp must be Greenwich mean time or local time.

When you use a time stamp as a search criteria, IXGBRWSE searches in the oldest-to-youngest direction, searching for a log block with an exactly matching time stamp. If no exact match is found, IXGBRWSE reads the next latest (youngest) time stamp. For information on how the SEARCH keyword works, see [z/OS MVS Programming: Assembler Services Guide](#).

The GMT parameter is required with the SEARCH parameter.

\, BUFFER=buffer

\, BUFFER64=buffer64

Specifies the name or address (using a register) of a required output field that contains the buffer into which the log block is read.

- BUFFER=buffer specifies that the location of the buffer is in 31-bit storage.
- BUFFER64=buffer64 specifies that the location of the buffer is in 64-bit storage.

The BUFFER and BUFFER64 parameters are mutually exclusive.

\, BUFFLEN=bufflen

Specifies the name or address (using a register) of a required 4-byte input field that contains the length of the buffer specified on the BUFFER or BUFFER64 parameter.

IXGBRWSE will return the length of the block in the BLKSIZE parameter, if specified. You can issue IXGBRWSE with BLKSIZE specified to obtain the length of the block and then re-issue IXGBRWSE using the returned BLKSIZE value in the BUFFLEN parameter.

\, ANSAREA=ansarea

Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

\, ANSLEN=anslen

Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 32 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

\, ANSLEN=anslen

Specifies the name (or register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 32 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area size, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

\, GMT=YES

\, GMT=NO

Specifies whether the time stamp specified on the SEARCH parameter is in Coordinated universal time (UTC) or local time.

- GMT=YES: The time stamp specified on the SEARCH parameter is in Greenwich mean time.
IXGBRWSE macro

- GMT=NO: The time stamp specified on the SEARCH parameter is local time.

BUFFKEY=buffkey
Specifies the name (or address in a register) of a 4-byte input field specifying
the storage key for the buffer specified on the BUFFER or BUFFER64 parameter.
If the caller is running in problem state, the caller’s PSW key and the key
specified in the BUFFKEY parameter must match.
If the caller is running in supervisor state, specify any syntactically valid (0
through 15) key on the BUFFKEY parameter.
If you omit the BUFFKEY parameter, the default used is the PSW key of the
caller.

BUFFALET=buffalet
Specifies the name (or address in a register) of a 4-byte input field specifying
the access list entry table (ALET) to be used to access the buffer specified on
the BUFFER or BUFFER64 keyword. If the buffer is ALET-qualified, the ALET
must index a valid entry on the task’s dispatchable unit access list (DUAL) or
specify a SCOPE=COMMON data space. An ALET that indexes the system
logger PASN-AL list will not work.
The default is 0, which means that the buffer is in the calling program’s
primary address space.

BLKSIZE=blksize
Specifies the name or address (using a register) of a 4-byte output field where
the actual size of the requested log block is returned.

RETBLOCKID=retblockid
Specifies the name or address (using a register) of an 8-byte output field where
the identifier of the requested log block is returned.

TIMESTAMP=timestamp
Specifies the name or address (using a register) of a 16-byte output field where
the Coordinated universal time and local time stamps associated with the
requested log block is returned. The UTC time stamp is first, then the local
time stamp. Both time stamps will be in TOD-clock format.

MODE=SYNC
MODE=SYNCECB
MODE=SYNCEXIT
Specifies that the request should be processed in one of the following ways:
- MODE=SYNC: Specifies that the request process synchronously. Control is
  not returned to the caller until request processing is complete. If necessary,
  the calling program will be suspended until the request completes.
- MODE=SYNCECB: Specifies that the request process synchronously if
  possible. If the request processes asynchronously, control returns to the caller
  before the request completes and the event control block (ECB) specified on
  the ECB parameter is posted when the request completes. The ECB
  parameter is required with MODE=SYNCECB.
- MODE=SYNCEXIT: Specifies that the request process synchronously, if
  possible. If the request cannot be processed synchronously, your complete
  exit (specified on the COMPLETEEXIT parameter on the IXGCONN request)
  gets control when this request completes. Control returns to the caller with a
  return and reason code indicating that the request is not complete. The
  system passes the data specified on the REQDATA parameter, if specified, to
  the complete exit.
**IXGBRWSE macro**

When a MODE=SYNCEXIT request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to RETBLOCKID and TIMESTAMP.

To use this parameter, the system where the application is running must be IPLed. The application must run in supervisor state, key 0-7 to use this parameter.

**ECB=**

Specifies the name or address (using a register) of a 4-byte input field that contains an event control block (ECB) to be posted when the request completes.

Before coding ECB, you must ensure that:

- You initialize the ECB to zero.
- The ECB must reside in either common storage or the home address space at the time the IXGBRWSE request is issued.
- The virtual storage area specified for the ECB must reside on a fullword boundary.

**,PLISTVER=IMPLIED_VERSION**

,PLISTVER=MAX

,PLISTVER=plistver

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - DIAG
  - REQDATA
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - MAXNUMLOGBLOCKS
  - MULTIBLOCK
  - RETBLOCKINFO

**To code:** Specify in this input parameter one of the following:

- **IMPLIED_VERSION**
- **MAX**
- A decimal value of 0, 1 or 2
IXGBRWSE macro

\texttt{,RETCODE=retcode}

Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

\texttt{,RSNCODE=rsncode}

Specifies a name or address (using a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

\texttt{,MF=S}

\texttt{,MF=(L,list addr)}

\texttt{,MF=(L,list addr,attr)}

\texttt{,MF=(L,list addr,0D)}

\texttt{,MF=(E,list addr)}

\texttt{,MF=(E,list addr,COMPLETE)}

\texttt{,MF=(E,list addr,NOCHECK)}

\texttt{,MF=(M,list addr)}

\texttt{,MF=(M,list addr,COMPLETE)}

\texttt{,MF=(M,list addr,NOCHECK)}

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

\textbf{IBM recommends} that you use the modify and execute forms in the following order:

- Use \texttt{MF=(M,list_addr,COMPLETE)}, specifying appropriate parameters, including all required ones.
- Use \texttt{MF=(M,list_addr,NOCHECK)}, specifying the parameters you want to change.
- Use \texttt{MF=(E,list_addr,NOCHECK)}, to execute the macro.

\texttt{,list addr}

The name of a storage area to contain the parameters.

\texttt{,attr}

An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.
IXGBRWSE macro

,,COMPLETE
   Specifies that the system is to check for required parameters and supply
defaults for omitted optional parameters.

,,NOCHECK
   Specifies that the system is not to check for required parameters and is not
to supply defaults for omitted optional parameters.

REQUEST=RESET option of IXGBRWSE
The IXGBRWSE macro with the REQUEST=RESET parameter allows a program to
re-position the browse cursor to either the youngest or oldest block in the log
stream.

Syntax for REQUEST=RESET
The IXGBRWSE REQUEST=RESET macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IXGBRWSE.</td>
</tr>
<tr>
<td>IXGBRWSE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IXGBRWSE.</td>
</tr>
<tr>
<td>REQUEST=RESET</td>
<td></td>
</tr>
<tr>
<td>,STREAMTOKEN=streamtoken</td>
<td>streamtoken: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BROWSETOKEN=browsetoken</td>
<td>browsetoken: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,POSITION=YOUNGEST</td>
<td></td>
</tr>
<tr>
<td>,POSITION=OLDEST</td>
<td></td>
</tr>
<tr>
<td>,ANSAREA=ansarea</td>
<td>ansarea: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ANSLEN=anslen</td>
<td>anslen: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>VIEW=ACTIVE</td>
<td></td>
</tr>
<tr>
<td>VIEW=ALL</td>
<td></td>
</tr>
<tr>
<td>MODE=SYNC</td>
<td>Default: MODE=SYNC</td>
</tr>
<tr>
<td>MODE=SYNCECB</td>
<td></td>
</tr>
<tr>
<td>MODE=SYNCEXIT</td>
<td></td>
</tr>
</tbody>
</table>
Parameters for REQUEST=RESET

The parameters are explained as follows:

**REQUEST=RESET**

Requests that the browse cursor be repositioned at either the oldest or youngest block in the log stream.

**STREAMTOKEN=streamtoken**

Specifies the name or address (using a register) of a required 16-byte input field containing the token for the log stream that you want to search. The stream token is returned by the IXGCONN service at connection to the log stream.

**BROWSETOKEN=browsetoken**

Specifies the name or address (using a register) of a required 4-byte input field containing the identifier for the browse session which was returned from the IXGBRWSE REQUEST=START request.

**POSITION=YOUNGEST, POSITION=OLDEST**

Specifies the cursor position desired, at either the youngest or the oldest log block in the log stream.

**ANSAREA=ansarea**

Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

---

**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,ECB=ecb</td>
<td>ecb: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=plistver</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
<tr>
<td>,MF=(M,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(M,list addr,COMPLETE)</td>
<td></td>
</tr>
<tr>
<td>,MF=(M,list addr,NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>
IXGBRWSE macro

\,\ANSLEN=anslen

Specifies the name (or address in a register) of the 4-byte field containing the
answer area length. The length of the answer area must be at least 40 bytes
and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the
ANSAA\_PREFERRED\_SIZE field of the IXGANSAA macro.

\,\ANSLEN=anslen

Specifies the name (or register) of the 4-byte field containing the answer area
length. The length of the answer area must be at least 32 bytes and must be
the same length as the field specified in ANSAREA.

To ascertain the optimal answer area size, look at the
ANSAA\_PREFERRED\_SIZE field of the IXGANSAA macro.

\,VIEW=ACTIVE
\,VIEW=ALL

Specifies whether requests issued during this browse session return active data
only, or both active and inactive data. Active data is data that has not been
marked for deletion via the IXGDELETE service. Inactive data is data that has
been deleted via IXGDELETE but has not been physically deleted from the log
stream because of the retention period specified in the log stream definition in
the LOGR couple data set.

- \,VIEW=ACTIVE, which is the default, specifies that in this browse session,
system logger will only return active data from the log stream.
- \,VIEW=ALL specifies that in this browse session, system logger will return
both active and inactive data.

When VIEW=ALL is specified and a log block is returned, system logger sets
a flag in the answer area, AnsaaBlkFromInactive, indicating whether the
block was active or eligible for deletion.

The system where IXGBRWSE is issued must be IPLed.

\,\MODE=SYNC
\,\MODE=SYNCECB
\,\MODE=SYNCEXIT

Specifies that the request should be processed in one of the following ways:

- \,MODE=SYNC: Specifies that the request process synchronously. Control is
not returned to the caller until request processing is complete. If necessary,
the calling program will be suspended until the request completes.
- \,MODE=SYNCECB: Specifies that the request process synchronously if
possible. If the request processes asynchronously, control returns to the caller
before the request completes and the event control block (ECB) specified on
the ECB parameter is posted when the request completes. The ECB
parameter is required with MODE=SYNCECB.
- \,MODE=SYNCEXIT: Specifies that the request process synchronously, if
possible. If the request cannot be processed synchronously, your complete
exit (specified on the COMPLETEEXIT parameter on the IXGCONN request)
gets control when this request completes. Control returns to the caller with a
return and reason code indicating that the request is not complete. The
system passes the data specified on the REQDATA parameter, if specified, to
the complete exit.

When a MODE=SYNCEXIT request processes asynchronously, system logger
maintains latent binds to the storage location specified by the answer area
(ANSAREA) fields, and, if specified, to RETBLOCKID and TIMESTAMP.
To use this parameter, the system where the application is running must be IPLed. The application must run in supervisor state, key 0-7 to use this parameter.

**ECB=ecb**

Specifies the name or address (using a register) of a 4-byte input field that contains an event control block (ECB) to be posted when the request completes.

Before coding ECB, you must ensure that:

- You initialize the ECB to zero.
- The ECB must reside in either common storage or the home address space at the time the IXGBRWSE request is issued.
- The virtual storage area specified for the ECB must reside on a fullword boundary.

**PLISTVER=**

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - DIAG
  - REQDATA
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - MAXNUMLOGBLOCKS
  - MULTIBLOCK
  - RETBLOCKINFO

**To code:** Specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1 or 2
IXGBRWSE macro

\texttt{,RETCODE=\texttt{retcode}}

Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

\texttt{,RSNCODE=\texttt{rsncode}}

Specifies a name or address (using a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

\texttt{,MF=S}

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

\textbf{IBM recommends} that you use the modify and execute forms in the following order:

- Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,list_addr,NOCHECK), to execute the macro.

\texttt{,list addr}

The name of a storage area to contain the parameters.

\texttt{,attr}

An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code \texttt{attr}, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.
### IXGRWSE macro

**COMPLETE**
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

**NOCHECK**
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

### REQUEST=END option of IXGRWSE

The IXGRWSE macro with the REQUEST=END parameter ends the browse session begun with the REQUEST=START parameter.

### Syntax for REQUEST=END

The IXGRWSE REQUEST=END macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IXGRWSE.</td>
</tr>
<tr>
<td>IXGRWSE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IXGRWSE.</td>
</tr>
<tr>
<td>REQUEST=END</td>
<td></td>
</tr>
<tr>
<td>,STREAMTOKEN=streamtoken</td>
<td>streamtoken: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BROWSETOKEN=browsetoken</td>
<td>browsetoken: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ANSAREA=ansarea</td>
<td>ansarea: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ANSLEN=anslen</td>
<td>anslen: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>MODE=SYNC</td>
<td>Default: MODE=SYNC</td>
</tr>
<tr>
<td>MODE=SYNCECB</td>
<td></td>
</tr>
<tr>
<td>MODE=SYNCEXIT</td>
<td></td>
</tr>
<tr>
<td>,ECB=ecb</td>
<td>ecb: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=plistver</td>
<td></td>
</tr>
</tbody>
</table>
### IXBRWSE macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr,NOCHECK)</td>
<td></td>
</tr>
<tr>
<td>,MF=(M,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(M,list addr,COMPLETE)</td>
<td></td>
</tr>
<tr>
<td>,MF=(M,list addr,NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters for REQUEST=END

The parameters are explained as follows:

#### REQUEST=END
Requests that the browse session be ended.

#### ,STREAMTOKEN=streamtoken
Specifies the name or address (using a register) of a required 16-byte input field containing the token for the log stream that you want to search. The stream token is returned by the IXGCONN service at connection to the log stream.

#### ,BROWSETOKEN=browsetoken
Specifies the name or address (using a register) of a required 4-byte input field containing the identifier for the browse session which was returned from the IXGBRWSE REQUEST=START request.

#### ,ANSAREA=ansarea
Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

#### ,ANSLEN=anslen
Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

#### ,MODE=SYNC
#### ,MODE=SYNCECB
#### ,MODE=SYNCEXIT
Specifies that the request should be processed in one of the following ways:
**IXGBRWSE macro**

- **MODE=SYNC**: Specifies that the request process synchronously. Control is not returned to the caller until request processing is complete. If necessary, the calling program will be suspended until the request completes.

- **MODE=SYNCECB**: Specifies that the request process synchronously if possible. If the request processes asynchronously, control returns to the caller before the request completes and the event control block (ECB) specified on the ECB parameter is posted when the request completes. The ECB parameter is required with MODE=SYNCECB.

  When a MODE=SYNCECB request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to BUFFER, BUFFER64, RETBLOCKID, TIMESTAMP, and BLKSIZE.

- **MODE=SYNCEXIT**: Specifies that the request process synchronously, if possible. If the request cannot be processed synchronously, your complete exit (specified on the COMPLETEEXIT parameter on the IXGCONN request) gets control when this request completes. Control returns to the caller with a return and reason code indicating that the request is not complete. The system passes the data specified on the REQDATA parameter, if specified, to the complete exit.

  When a MODE=SYNCEXIT request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to BUFFER, BUFFER64, RETBLOCKID, TIMESTAMP, and BLKSIZE.

  To use this parameter, the system where the application is running must be IPLed. The application must run in supervisor state, key 0-7 to use this parameter.

**ECB=ecb**

Specifies the name or address (using a register) of a 4-byte input field that contains an event control block (ECB) to be posted when the request completes.

Before coding ECB, you must ensure that:

- You initialize the ECB to zero.
- The ECB must reside in either common storage or the home address space at the time the IXGBRWSE request is issued.
- The virtual storage area specified for the ECB must reside on a fullword boundary.

**PLISTVER=IMPLIED_VERSION, PLISTVER=MAX, PLISTVER=plistver**

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX
IXGBRWSE macro

ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- 0, supports all parameters except those specifically referenced in higher versions.
- 1, supports both the following parameters and parameters from version 0:
  - DIAG
  - REQDATA
- 2, supports both the following parameters and parameters from version 0 and 1:
  - MAXNUMLOGBLOCKS
  - MULTIBLOCK
  - RETBLOCKINFO

To code: Specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1 or 2

`,RETCODE`=retcode
Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

`,RSNCODE`=rsncode
Specifies a name or address (using a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

`,MF=S
 `,MF=`(L,list addr)
 `,MF=`(L,list addr,attr)
 `,MF=`(L,list addr,0D)
 `,MF=`(E,list addr)
 `,MF=`(E,list addr,COMPLETE)
 `,MF=`(E,list addr,NOCHECK)
 `,MF=`(M,list addr)
 `,MF=`(M,list addr,COMPLETE)
 `,MF=`(M,list addr,NOCHECK)

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.
Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:

- Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,list_addr,NOCHECK), to execute the macro.

`,list_addr`
The name of a storage area to contain the parameters.

`,attr`
An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code `attr`, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

`,COMPLETE`
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

`,NOCHECK`
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

**ABEND codes**

The IXGBRWSE service may issue abend X'1C5' with reason codes X'804', X'85F' or X'30006'. See [z/OS MVS System Codes](https://www.ibm.com/support/knowledgecenter/en/SS752_1.4.1/ixgans00.htm) for more information on this abend.

**Return and reason codes**

When IXGBRWSE macro returns control to your program, GPR 15 contains a return code and GPR 0 contains a reason code.

**Note:** A program invoking the IXGBRWSE service may indicate via the MODE parameter that requests which can not be completed synchronously should have control returned to the caller prior to completion of the request. When the request does complete, the invoker will be notified and the return and reason codes are in the answer area mapped by IXGANSAA.

The IXGCON mapping macro provides equate symbols for the return and reason codes. The equate symbols associated with each hexadecimal return code are as follows:

- 00 IXGRSNCODEOK - Service completes successfully.
- 04 IXGRSNCODEWARNING - Service completes with a warning.
- 08 IXGRETCODEERROR - Service does not complete.
- 0C IXGRETCODECOMPERROR - Service does not complete.
The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00          | xxxx0000    | **Equate Symbol:** IxgRsnCodeOk  
**Explanation:** Request processed successfully. |
| 04          | xxxx0401    | **Equate Symbol:** IxgRsnCodeProcessedAsynch  
**Explanation:** Program error. The program specified MODE=SYNCECB and the request must be processed asynchronously.  
**Action:** Wait for the ECB specified on the ECB parameter to be posted, indicating that the request is complete. Check the ANSAA_ASYNCH_RETCODE and ANSAA_ASYNCH_RSNCODE fields, mapped by IXGANSSAA, to determine whether the request completed successfully. |
| 04          | xxxx0402    | **Equate Symbol:** IxgRsnCodeWarningDel  
**Explanation:** Environment error. The request completed successfully, but the data requested was deleted from the log stream. The next available data in the log stream in the direction specified is returned.  
**Action:** Determine whether this is an acceptable condition for your application. If so, ignore this condition. If not, provide serialization or some other installation protocol to prevent deletes from being performed by other applications on the log stream during a browse session. |
| 04          | xxxx0403    | **Equate Symbol:** IxgRsnCodeWarningGap  
**Explanation:** Environment error. The request completed successfully, but the data requested was unreadable. The next readable data in the log stream in the specified direction is returned. This condition could be caused by either an I/O error while attempting to read a log data set or a log data set deleted without using the logger interfaces.  
**Action:** The action necessary is completely up to the application, depending on how critical your data is. You can do one of the following:  
• Accept this condition and continue reading.  
• Stop processing the log all together.  
• Attempt to get the problem rectified, if possible, and then attempt to re-read the log data. |
### Table 117. Return and Reason Codes for the IXGBRWSE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04          | xxxx0405    | **Equate Symbol**: IxgRsnCodeWarningLossOfData  
**Explanation**: Environment error. Returned for READCURSOR, START OLDEST and RESET OLDEST requests. This condition occurs when a system and coupling facility fail and not all of the log data in the log stream could be recovered.  
- For READCURSOR: A log block has been returned, but there may be log blocks permanently missing between this log block and the one previously returned.  
- For START OLDEST and RESET OLDEST: The oldest log blocks in the log stream may be permanently missing, the browse cursor is set at the oldest available log block.  
**Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss. |
| 04          | xxxx0416    | **Equate Symbol**: IxgRsnCodeWarningMultiblock  
**Explanation**: Environment error. Returned for READCURSOR requests with MULTIBLOCK=YES specified only. The request completed successfully, which means that some log block data was returned, but at least one of the log blocks returned in the buffer area encountered a warning return code condition. To determine which log block or blocks encountered the warning condition, check the fields, Ixgbrmlt_RetCode and Ixgbrmlt_RsnCode, as the log blocks are processed by your program.  
**Action**: The action necessary is completely up to the application, depending on how critical your data is. You can do one of the following:  
- Accept this condition and continue reading.  
- Stop processing the log all together.  
- Attempt to get the problem rectified, if possible, and then attempt to re-read the log data. |
### IXGBRWSE macro

**Table 117. Return and Reason Codes for the IXGBRWSE Macro (continued)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04          | xxxx0417    | **Equate Symbol**: IxgRsnCodeMultiblockErrorWarning  
**Explanation**: Environment error. Returned for READCURSOR requests with MULTIBLOCK=YES specified only. A log block has been returned, but an error condition was encountered while attempting to read more data. This may be issued when some log block data is returned and an end of the log stream (eol) is reached.  
**Action**: The action necessary is completely up to the application, depending on how critical your data is. You can do one of the following:  
- Accept this condition and continue reading.  
- Stop processing the log all together.  
- Attempt to get the problem rectified, if possible, and then attempt to re-read the log data. |
| 08          | xxxx0801    | **Equate Symbol**: IxgRsnCodeBadParmlist  
**Explanation**: Program error. The parameter list could not be accessed.  
**Action**: Ensure that the storage area for the parameter list is accessible to the system logger for the duration of the request. The parameter list storage must be addressable in the caller’s primary address space and in the same key as the caller. |
| 08          | xxxx0802    | **Equate Symbol**: IxgRsnCodeXESError  
**Explanation**: System error. A severe cross-system extended services (XES) error has occurred.  
**Action**: See ANSAA_DIAG1 for the XES return code and ANSAA_DIAG2 for the XES reason code. |
| 08          | xxxx0803    | **Equate Symbol**: IxgRsnCodeBadBuffer  
**Explanation**: Program error. The virtual storage area specified on the BUFFER or BUFFER64 parameter is not addressable. On IXGBRWSE READCURSOR MULTIBLOCK requests, the buffer address must be on a word boundary.  
**Action**: Ensure that the storage area specified on the BUFFER or BUFFER64 parameter is accessible to system logger for the duration of the request. If the BUFFKEY parameter is specified, make sure it contains a valid key associated with the storage area. If BUFFKEY is not used, ensure that the storage is in the same key as the program at the time the logger service was requested. The storage must be addressable in the caller’s primary address space. For IXGBRWSE READCURSOR MULTIBLOCK requests, put the buffer address on a word boundary. |
## IXGBRWSE macro

**Table 117. Return and Reason Codes for the IXGBRWSE Macro (continued)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0804    | Equate Symbol: IxgRsnCodeNoBlock  
**Explanation**: Program error. The block identifier or time stamp does not exist in the requested view of the log stream. If the SEARCH parameter was specified on a START request, the time stamp is greater than any block in the log stream. Either the value provided was never a valid location within the log stream, or a prior IXGDELET request deleted the portion of the log stream it referred to.  
**Action**: Ensure that the value provided references an existing portion of the log stream. |
| 08          | xxxx0806    | Equate Symbol: IxgRsnCodeBadStmToken  
**Explanation**: Program error. One of the following occurred:  
• The stream token was not valid.  
• The specified request was issued from an address space other than the connector’s address space.  
**Action**: Do one of the following:  
• Make sure that the stream token specified is valid.  
• Ensure that the request was issued from the connector’s address space. |
| 08          | xxxx0807    | Equate Symbol: IxgRsnCodeBadBrwToken  
**Explanation**: Program error. The browse token specified is not valid.  
**Action**: Ensure that the browse token being passed to the IXGBRWSE service is the same one returned from the IXGBRWSE REQUEST=START function. |
| 08          | xxxx080A    | Equate Symbol: IxgRsnCodeRequestLocked  
**Explanation**: Program error. The program issuing the request is holding a lock.  
**Action**: Ensure that the program issuing the request is not holding a lock. |
| 08          | xxxx080F    | Equate Symbol: IxgRsnCodeBadBufsize  
**Explanation**: Program error. The buffer specified on the BUFFER or BUFFER64 parameter is not large enough to contain the next log block. No data is returned.  
**Action**: Obtain a buffer of at least the length returned in the BLKSIZE parameter and then re-issue the request. |
### IXGBRWSE Macro

**Table 117. Return and Reason Codes for the IXGBRWSE Macro (continued)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0814    | **Equate Symbol**: IxgRsnCodeNotAvailForIPL  
**Explanation**: Environment error. The system logger address space is not available for the remainder of this IPL. The system issues messages about this error during system logger initialization.  
**Action**: See the explanation for system messages issued during system logger initialization. |
| 08          | xxxx0815    | **Equate Symbol**: IxgRsnCodeNotEnabled  
**Explanation**: Program error. The program issuing the request is not enabled for I/O and external interrupts, so the request fails.  
**Action**: Make sure the program issuing the request is enabled for I/O and external interrupts. |
| 08          | xxxx0816    | **Equate Symbol**: IxgRsnCodeBadAnslen  
**Explanation**: Program error. The answer area length (ANSLEN parameter) is not large enough. The system logger returned the required size in the Ansaa_Preferred_Size field of the answer area, mapped by IXGANSAA macro.  
**Action**: Re-issue the request, specifying an answer area of the required size. |
| 08          | xxxx0817    | **Equate Symbol**: IxgRsnCodeBadAnsarea  
**Explanation**: Program error. The storage area specified on the ANSAREA parameter cannot be accessed. This may occur after the system logger address space has terminated.  
**Action**: Specify storage that is in the caller’s primary address space and in the same key as the calling program at the time the system logger service was issued. This storage must be accessible until the request completes. |
| 08          | xxxx0818    | **Equate Symbol**: IxgRsnCodeBadBlockidStor  
**Explanation**: Program error. The storage area specified by BLOCKID cannot be accessed.  
**Action**: Ensure that the storage area is accessible to system logger for the duration of the request. The storage must be addressable in the caller’s primary address space and in the same key as the caller. |
| 08          | xxxx082D    | **Equate Symbol**: IxgRsnCodeExpiredStmToken  
**Explanation**: Environment error. The stream token is no longer valid because the connector has been disconnected.  
**Action**: Connect to the log stream again before issuing any functional requests. |
### IXGRWSE macro

Table 117. Return and Reason Codes for the IXGRWSE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0836    | **Equate Symbol**: IxgRsnCodeBadGap  
**Explanation**: Environment error. The request failed because the requested log data was unreadable. This condition could be caused by either an I/O error while attempting to read a log data set or a log data set deleted without using logger interfaces.  
**Action**: For an IXGRWSE request, choose on of the following:  
- Continue processing.  
- Stop processing the log stream all together.  
- Attempt to get the problem rectified if possible, then attempt to re-read the log data.  
For an IXGDELET request, the block identifier of the first accessible block toward the youngest data in the log stream is returned in the ANSAA_GAPS_NEXT_BLKID field in the answer area mapped by the IXGANSFA macro. If appropriate, re-issue the IXGDELET request using this block identifier. |
| 08          | xxxx0837    | **Equate Symbol**: IxgRsnCodeBadTimestamp  
**Explanation**: Program error. The storage area specified by TIMESTAMP cannot be accessed.  
**Action**: Ensure that the storage area is accessible to the system logger service for the duration of the request. The storage must be addressable in the caller's primary address space and in the same key as the caller. |
| 08          | xxxx083B    | **Equate Symbol**: IxgRsnCodeBadBTokenStor  
**Explanation**: Program error. The storage area specified by BROWSETOKEN cannot be accessed.  
**Action**: Ensure that the storage area is accessible to the system logger for the duration of the request. The storage must be addressable in the caller's primary address space and in the same key as the caller. |
| 08          | xxxx083D    | **Equate Symbol**: IxgRsnCodeBadECBStor  
**Explanation**: Program error. The ECB storage area was not accessible to the system logger.  
**Action**: Ensure that the storage area is accessible to the system logger for the duration of the request. The storage must be addressable in the caller's home address space and in the same key as the caller. |
### Return and Reason Codes for the IXGBRWSE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx083F     | **Equate Symbol**: IxgRsnCodeTestartError  
**Explanation**: System error. An unexpected error was encountered while attempting to validate the buffer ALET.  
**Action**: See ANSAA_DIAG1 in the answer area mapped by the IXGANSAA macro for the return code from the TESTART system service. |
| 08          | xxxx0841     | **Equate Symbol**: IxgRsnCodeBadBufferAlet  
**Explanation**: Program error. The buffer ALET specified is not zero and does not represent a valid entry on the caller’s dispatchable unit access list (DUAL). See the ANSAA_DIAG1 field of the answer area, mapped by the IXGANSAA macro, for the return code from the TESTART system service.  
**Action**: Ensure that the correct ALET was specified. If not, provide the correct ALET. Otherwise, add the correct ALET to dispatchable unit access list (DUAL). |
| 08          | xxxx0845     | **Equate Symbol**: IxgRsnCodeInvalidFunc  
**Explanation**: System error. One of 2 problems was detected.  
1. The parameter list for this service contains an unrecognizable function code. The parameter list storage may have been overlayed.  
2. The IXGBRWSE START is rejected because either:  
   • A: An unauthorized caller attempted to start a session when 100 or more browse sessions already exist for this connection. Or,  
   • B: An unauthorized caller attempted to start a session when 20 or more browse sessions already exist that show no recent activity. (An unauthorized caller is a caller whose PSW Key is >= 8 and that is not in supervisor state).  
For Case 2: DIAG1 in the Answer Area will contain 1 if ‘A’ is the case, and 2 if ‘B’ is the case.  
DIAG2 will contain the number of browse sessions that was exceeded.  
**Action**: Fix the problem and then re-issue the request. It may be necessary to terminate some Browse sessions that are not being used. |
| 08          | xxxx0846     | **Equate Symbol**: IxgRsnCodeEmptyStream  
**Explanation**: Environment error. The log stream is empty.  
**Action**: Wait for data to be written to the log stream before browsing for data. |
### Table 117. Return and Reason Codes for the IXGBRWSE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>xxxx0847</td>
<td><strong>Equate Symbol</strong>: IxgRsnCodeEOFDelete</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation</strong>: Environment error. The request prematurely reached the beginning or the end of the log stream. The portion of the log stream from the requested log data to either the beginning or the end of the log stream (depending on the direction of the read) was deleted from the log stream.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Determine whether this is an acceptable condition for your application. If so, ignore this condition. If not, provide serialization on the log stream or some other installation protocol to prevent deletes from being performed by other applications during a browse session.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0848</td>
<td><strong>Equate Symbol</strong>: IxgRsnCodeEndReached</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation</strong>: Environment error. The request failed and no log data is returned. For a READCURSOR request, the end of the log stream has been reached in the direction of the read. If the SEARCH parameter was specified on a READBLOCK request, the time stamp is greater than any block in the log stream.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: For the READCURSOR case, no more data exists in the log stream in the direction of the read. You can choose to stop reading, wait for more data to be written, or change the direction of the read. In the case where the SEARCH parameter was provided, ensure that the time stamp is less than or equal to the highest time stamp of a log block in the log stream.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0849</td>
<td><strong>Equate Symbol</strong>: IxgRsnCodeBadBuffkey</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation</strong>: Program error. The buffer key specified on the BUFFKEY parameter specifies an invalid key. Either the key is greater than 15 or the program is running in problem state and the specified key is not the same key as the PSW key at the time the system logger service was issued.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: For problem state programs, either do not specify the BUFFKEY parameter or else specify the same key as the PSW key at the time the system logger service was issued. For supervisor state programs, specify a valid storage key (0 &lt;= key &lt;= 15).</td>
</tr>
</tbody>
</table>
### Table 117. Return and Reason Codes for the IXGBRWSE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>xxxx084A</td>
<td><strong>Equate Symbol:</strong> IxgRsnCodeEOFGap</td>
</tr>
<tr>
<td></td>
<td><strong>Explanation:</strong> Environment error. The request prematurely reached the beginning or the end of the log stream. The portion of the log stream from the requested log data to either the beginning or the end of the log stream (depending on the direction of the read) was unreadable. This condition may be caused by either an I/O error while trying to read a log data set, or a log data set deleted without using logger interfaces.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> The action necessary is completely up to the application depending on how critical your data is. You can do one of the following:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Accept this condition and continue reading.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Stop processing the log all together.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Attempt to get the problem rectified, if possible, and then attempt to re-issue the request.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx084B</td>
<td><strong>Equate Symbol:</strong> IxgRsncodeLossOfDataGap</td>
</tr>
<tr>
<td></td>
<td><strong>Explanation:</strong> Environment error. The requested log data referenced a section of the log stream where log data is permanently missing. This condition occurs when a system or coupling facility is in recovery due to a failure, but not all of the log data in the log stream could be recovered.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>xxxx084D</td>
<td><strong>Equate Symbol:</strong> IxgRsnCodeLossOfDataEOF</td>
</tr>
<tr>
<td></td>
<td><strong>Explanation:</strong> Environment error. The request prematurely reached the beginning or the end of the log stream. The portion of the log stream from the requested log data to either the beginning or the end of the log stream (depending on direction of the read) was permanently lost. This condition occurs when a system or coupling facility is in recovery due to a failure, but not all of the log data in the log stream could be recovered.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Action:</strong> If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.</td>
<td></td>
</tr>
</tbody>
</table>
### IXGBRWSE Macro

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0852    | Equate Symbol: IxgRsnCodeBadBlkSizeStor  
Explanation: Program error. The storage area specified on the BLKSIZE parameter cannot be accessed.  
Action: Ensure that the storage area is accessible to system logger for the duration of the request. |
| 08          | xxxx085F    | Equate Symbol: IxgRsnPercToRequestor  
Explanation: Environment error. Percolation to the service requestor's task occurred because of an abend during system logger processing. Retry was not allowed.  
Action: Issue the request again. If the problem persists, contact the IBM Support Center. |
| 08          | xxxx0861    | Equate Symbol: IxgRsnCodeRebuildInProgress  
Explanation: Environment error. No requests can be processed for this log stream because a coupling facility structure re-build is in progress for the structure associated with this log stream.  
Action: Listen for ENF signal 48 that will indicate one of the following:  
• The log stream is available because the re-build completed successfully. Re-issue the request.  
• The re-build failed and the log stream is not available. |
| 08          | xxxx0862    | Equate Symbol: IxgRsnCodeXESPurge  
Explanation: Environment error. An cross-system extended services (XES) request has been purged due to re-build processing.  
Action: Listen for ENF signal 48 that will indicate one of the following:  
• The log stream is available because the re-build completed successfully. Re-issue the request.  
• The re-build failed and the log stream is not available. |
| 08          | xxxx0863    | Equate Symbol: IxgRsnCodeStructureFailed  
Explanation: Environment error. Either the coupling facility structure associated with the log stream has failed or the coupling facility itself has failed.  
Action: Listen for ENF signal 48 that will indicate one of the following:  
• The log stream is available because the re-build completed successfully. Re-issue the request.  
• The re-build failed and the log stream is not available. |
### Table 117. Return and Reason Codes for the IXGBRWSE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>xxxx0864</td>
<td><strong>Equate Symbol:</strong> IxgRsnCodeNoConnectivity</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation:</strong> Environment error. No connectivity exists to the coupling facility associated with the log stream. The system logger will either attempt to re-build the log stream in another coupling facility or the log stream will be disconnected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Listen for ENF signal 48 that will indicate one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The log stream is available because the re-build completed successfully. Re-issue the request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The re-build failed and the log stream is not available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The log stream has been disconnected from this system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If a re-build initiated because of a loss of connectivity previously failed, an ENF corresponding to this reason code might not be issued. Further action by the installation might be necessary to cause the change of the log stream status again. Check the log for messages IXG101I, IXG107I and related rebuild messages for information on resolving any outstanding issues.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0890</td>
<td><strong>Equate Symbol:</strong> IxgRsnCodeAddrSpaceNotAvail</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation:</strong> System error. The system logger address space failed and is not available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Do not issue system logger requests.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx0891</td>
<td><strong>Equate Symbol:</strong> IxgRsnCodeAddrSpaceInitializing</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation:</strong> System error. The system logger address space is not available because it is IPLing.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Listen for ENF signal 48, which will indicate when the system logger address space is available. Re-connect to the log stream, then re-issue this request. You can also listen for ENF signal 48, which will indicate if the system logger address space will not be available for the life of the IPL. In that case, do not issue system logger services.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx08D0</td>
<td><strong>Equate Symbol:</strong> IxgRsnCodeProblemState</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation:</strong> Environment error. The request was rejected because of one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The request was issued in SRB mode while the requestor was in problem program state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The SYNCEXIT parameter was specified while the requestor's PSW key was in problem program key.</td>
</tr>
</tbody>
</table>
|             |             | **Action:** Change the invoking environment to supervisor state.
### IXGBRWSE macro

#### Table 117. Return and Reason Codes for the IXGBRWSE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx08D1    | **Equate Symbol: IxgRsnCodeProgramKey**  
**Explanation:** Environment error. The request was rejected because of one of the following:  
* The request was issued in SRB mode while the requestor was in problem program key (key 8-F).  
* The SYNCEXIT parameter was specified while the requestor’s PSW key was in problem program key.  
**Action:** Change the invoking environment to a system key (key 0-7). |
| 08          | xxxx08D2    | **Equate Symbol: IxgRsnCodeNoCompleteExit**  
**Explanation:** Program error. MODE=SYNCEXIT was specified, but the connection request did not identify a complete exit.  
**Action:** Either change this request to a different MODE option, or reconnect to the log stream with a complete exit on the COMPLETEXIT parameter. |
| 08          | xxxx08D3    | **Equate Symbol: IxgRsnCodeFuncNotSupported**  
**Explanation:** Environment error. The options specified on the IXGBRWSE request are not supported on this system/maintenance level of system logger.  
**Action:** Either install the level of system logger that provides the support for the requested function, or do not specify options that are not supported at this level. |
| 0C          | xxxx0000    | **Equate Symbol: IxgRetCodeCompError**  
**Explanation:** User or System error. One of the following occurred:  
* You issued the FORCE IXGLOGR,ARM command to terminate the system logger address space.  
* System logger component error occurred.  
**Action:** If this reason code is not the result of forcing the system logger address space, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. Provide the diagnostic data in the answer area (IXGANSSAA) and any dumps or LOGREC entries from system logger. |

#### Example 1

Issue IXGBRWSE REQUEST=START to start a browse session, starting the browse cursor at the log block with the specified local time.

```
IXGBRWSE REQUEST=START,  
STREAMTOKEN=TOKEN,  
SEARCH=SRCHTIME,  
GMT=NO,  
BROWSETOKEN=BRSTOKEN,
```
Example 2

Issue IXGBRWSE REQUEST=READCURSOR to read the next consecutive log block in the specified direction. In this example, the default of MULTIBLOCK=NO has been taken.

```assembly
IXGBRWSE REQUEST=READCURSOR,
STREAMTOKEN=TOKEN,
BUFFER=BUFF,
BUFFLEN=BUFFLEN,
BUFFALET=ALET,
BLKSIZE=BLKSIZE,
DIRECTION=OLDTOYOUNG,
RETBLOCKID=RETBLK,
TIMESTAMP=TIMESTAMP,
BROWSETOKEN=BRSTOKEN,
MODE=SYNC,
ANSAREA=ANSAREA,
ANSLEN=ANSLEN,
RSNCODE=RSNCODE,
MF=S,
RETCODE=RETCODE
```

Example 3

Issue IXGBRWSE REQUEST=READBLOCK to read a log block selected by block identifier.

```assembly
IXGBRWSE REQUEST=READBLOCK,
STREAMTOKEN=TOKEN,
BLOCKID=BLKID,
BUFFER=BUFF,
BUFFLEN=BUFFLEN,
BUFFALET=ALET,
BLKSIZE=BLKSIZE,
RETBLOCKID=RETBLK,
TIMESTAMP=TIMESTAMP,
```

Example 2

Issue IXGBRWSE REQUEST=READCURSOR to read the next consecutive log block in the specified direction. In this example, the default of MULTIBLOCK=NO has been taken.

```assembly
IXGBRWSE REQUEST=READCURSOR,
STREAMTOKEN=TOKEN,
BUFFER=BUFF,
BUFFLEN=BUFFLEN,
BUFFALET=ALET,
BLKSIZE=BLKSIZE,
DIRECTION=OLDTOYOUNG,
RETBLOCKID=RETBLK,
TIMESTAMP=TIMESTAMP,
BROWSETOKEN=BRSTOKEN,
MODE=SYNC,
ANSAREA=ANSAREA,
ANSLEN=ANSLEN,
RSNCODE=RSNCODE,
MF=S,
RETCODE=RETCODE
```

Example 3

Issue IXGBRWSE REQUEST=READBLOCK to read a log block selected by block identifier.

```assembly
IXGBRWSE REQUEST=READBLOCK,
STREAMTOKEN=TOKEN,
BUFFER=BUFF,
BUFFLEN=BUFFLEN,
BUFFALET=ALET,
BLKSIZE=BLKSIZE,
RETBLOCKID=RETBLK,
TIMESTAMP=TIMESTAMP,
```

Example 2

Issue IXGBRWSE REQUEST=READCURSOR to read the next consecutive log block in the specified direction. In this example, the default of MULTIBLOCK=NO has been taken.

```assembly
IXGBRWSE REQUEST=READCURSOR,
STREAMTOKEN=TOKEN,
BUFFER=BUFF,
BUFFLEN=BUFFLEN,
BUFFALET=ALET,
BLKSIZE=BLKSIZE,
DIRECTION=OLDTOYOUNG,
RETBLOCKID=RETBLK,
TIMESTAMP=TIMESTAMP,
BROWSETOKEN=BRSTOKEN,
MODE=SYNC,
ANSAREA=ANSAREA,
ANSLEN=ANSLEN,
RSNCODE=RSNCODE,
MF=S,
RETCODE=RETCODE
```

Example 3

Issue IXGBRWSE REQUEST=READBLOCK to read a log block selected by block identifier.

```assembly
IXGBRWSE REQUEST=READBLOCK,
STREAMTOKEN=TOKEN,
BUFFER=BUFF,
BUFFLEN=BUFFLEN,
BUFFALET=ALET,
BLKSIZE=BLKSIZE,
RETBLOCKID=RETBLK,
TIMESTAMP=TIMESTAMP,
```
IXGBRWSE macro

```
BROWSETOKEN=BRSTOKEN,          X
MODE=SYNC,                      X
ANSAREA=ANSAREA,                X
ANSLEN=ANSLEN,                  X
RSNCODE=RSNCODE,                X
MF=S,                           X
RETCODE=RETCODE

ANSLEN  DC A(L'ANSAREA)        length of logger's answer area
BUFSIZE  DC F'200'             buffer length
TOKEN    DS CL16               stream token from connect
BRSTOKEN DS CL4                returned browse token
BUFF     DS CL200              buffer where data will be put
ALET     DS F'1'               buffer alet in secondary
BLKSIZE  DS F                 block size of buffer
RETBLK   DS CL8               return block id
BCKID    DS CL8               specific block id to browse
TIMESTAMP DS CL16             returned time stamp stick format
ANSAREA  DS CL(ANSAA_LEN)      answer area for log requests
RETCODE  DS F                 return code
RSNCODE  DS F                 reason code
DATAREA  DSECT
         IXGANSA LIST=YES        answer area
```

Example 4

Issue IXGBRWSE REQUEST=RESET to reset the cursor at the youngest block in the log stream.

```
IXGBRWSE REQUEST=RESET,         X
STREAMTOKEN=_TOKEN,             X
POSITION=YOUNGEST,              X
BROWSETOKEN=BRSTOKEN,           X
MODE=SYNC,                      X
ANSAREA=ANSAREA,                X
ANSLEN=ANSLEN,                  X
RSNCODE=RSNCODE,                X
MF=S,                           X
RETCODE=RETCODE

ANSLEN  DC A(L'ANSAREA)        length of logger's answer area
TOKEN    DS CL16               stream token from connect
BRSTOKEN DS CL4                returned browse token
ANSAREA  DS CL(ANSAA_LEN)      answer area for log requests
RETCODE  DS F                 return code
RSNCODE  DS F                 reason code
DATAREA  DSECT
         IXGANSA LIST=YES        answer area
```

Example 5

Issue IXGBRWSE REQUEST=END to end a browse session.

```
IXGBRWSE REQUEST=END,          X
STREAMTOKEN=TOKEN,             X
BROWSETOKEN=BRSTOKEN,          X
MODE=SYNC,                      X
ANSAREA=ANSAREA,                X
ANSLEN=ANSLEN,                  X
RSNCODE=RSNCODE,                X
MF=S,                           X
RETCODE=RETCODE

ANSLEN  DC A(L'ANSAREA)        length of logger's answer area
TOKEN    DS CL16               stream token from connect
BRSTOKEN DS CL4                browse token from browse start
ANSAREA  DS CL(ANSAA_LEN)      answer area for log requests
RETCODE  DS F                 return code
RSNCODE  DS F                 reason code
DATAREA  DSECT
         IXGANSA LIST=YES        answer area
```
**Example 6**

Issue IXGBRWSE REQUEST=END to end a browse session asynchronously, if synchronous processing is not possible.

```assembly
IXGBRWSE REQUEST=END, X
STREAMTOKEN=TOKEN, X
BROWSETOKEN=BRSTOKEN, X
MODE=SYNCECB, X
ECB=ANELCB, X
ANSAREA=ANSAREA, X
ANSLEN=ANSLEN, X
RSNCODE=RSNCODE, X
MF=S, X
RETCODE=RETCODE
```

* `*` if rsncode = '00000401'X then wait on
  * the ecb ANECB.

```assembly
ANSLEN DC A(L'ANSAREA) length of logger's answer area
TOKEN DS CL16 stream token from connect
BRSTOKEN DS CL4 browse token from browse start
ANSAREA DS CL(ANSAA_LEN) answer area for log requests
ANELCB DS F ecb on which to wait
RETCODE DS F return code
RSNCODE DS F reason code
DATAREA DSECT
  IXGANSAA LIST=YES answer area
```

**Example 7**

Issue IXGBRWSE REQUEST=END using registers.

```assembly
LA R6,TOKEN place stream token in reg 6
IXGBRWSE REQUEST=END, X
STREAMTOKEN=(6), X
BROWSETOKEN=BRSTOKEN, X
MODE=SYNC, X
ANSLEN=ANSLEN, X
ANSAREA=ANSAREA, X
RSNCODE=RSNCODE, X
MF=S, X
RETCODE=RETCODE
```

```assembly
ANSLEN DC A(L'ANSAREA) length of logger's answer area
TOKEN DS CL16 stream token from connect
BRSTOKEN DS CL4 browse token from browse start
ANSAREA DS CL(ANSAA_LEN) answer area for log requests
RETCODE DS F return code
RSNCODE DS F reason code
DATAREA DSECT
  IXGANSAA LIST=YES answer area
R6 EQU 6
```
Chapter 132. IXGCONN — Connect/disconnect to log stream

Description

Use the IXGCONN macro to connect a program to a specific log stream or disconnect a program from a specific log stream.

IXGCONN returns a unique connection identifier called a stream token on completion of the IXGCONN REQUEST=CONNECT request. Subsequent logger services use the stream token to identify the connection. If multiple applications connect to the same log stream, the log blocks written from the different applications are merged.

The IXGCONN connect service can be used in the following ways:
• Once a program has connected to a log stream, any application running in the same address space shares the connect status and may share the same stream token to issue other logger services. Any program in the address space can disconnect the entire address space from the log stream by issuing the IXGCONN REQUEST=DISCONNECT service.
• Multiple programs in a single address space can issue IXGCONN REQUEST=CONNECT individually to connect to the same log stream and receive separate stream tokens. Each program must disconnect from the log stream individually.
• Multiple address spaces on one or more MVS systems may connect to a single log stream, but each one must issue IXGCONN individually to connect and then disconnect from the log stream. Each one receives a unique stream token; address spaces cannot share a stream token.

Note that a DASD-only log stream is single-system in scope. This means that only one system may connect to a DASD-only log stream, although there can be multiple connections from that one system.

The parameter descriptions indicate parameters that can only be used in supervisor state, PSW key zero. All others can be used in problem or supervisor state with any PSW key.

For information about using the system logger services and the IXGCONN request, see z/OS MVS Programming: Assembler Services Guide, which includes information about related macros IXGBRWSE, IXGDELET, IXGWRITE, IXGINVNT, and IXGQUERY.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization</td>
<td>Problem state with any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN=HASN, any SASN</td>
</tr>
<tr>
<td>AMODE</td>
<td>31-bit or 64-bit</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks may be held.</td>
</tr>
</tbody>
</table>
Environmental factor
Control parameters: None.

Programming requirements
- The parameter list for this service must be addressable in the caller's primary address space.
- Include the IXGCON mapping macro in your program. This macro provides a list of equate symbols for the system logger services.
- Include mapping macro IXGANSAA in your program. This macro shows the format of the answer area output returned for each system logger service in the ANSAREA parameter.
- If you use IXGCONN REQUEST=CONNECT,...,MF=(E,parmlist,NOCHECK) with either the STREAMTOKEN=xxxx or the USERDATA=yyyy keyword, the following procedure must be followed. When the processing is complete, move the STREAMTOKEN or USERDATA values from the parameter list specified on MF= to your own storage.
- Each task that issues IXGCONN REQUEST=CONNECT to connect to a log stream must later issue IXGCONN REQUEST=DISCONNECT to disconnect from the log stream. When a task disconnects from the log stream, the stream token that identified the connection expires. Any requests that use the stream token after the disconnect are rejected with reason code X'82D'.
- If a task that issued the IXGCONN REQUEST=CONNECT request ends before issuing a disconnect request, system logger automatically disconnects the task from the log stream. This means that the unique log stream connection identifier, or the STREAMTOKEN, is no longer valid. The application receives an expired log stream token error response with reason code X'82D', if this application continues to use the same STREAMTOKEN after the task has been disconnected on subsequent logger service requests.
- Any job step task (JST) terminates within the address space that has a connection to the log stream. System logger treats any job step task termination in a manner similar to an address space termination. That is, all log stream connections are disconnected and logger associations are terminated with the address space.

If this condition occurs and there remains an expected use of a log stream, then a new log stream connection will be required.

Restrictions
- All storage areas specified in this service must be in the same storage key as the caller's storage key and must exist in the caller's primary address space.
- The caller cannot have an EUT FRR established.
- If the Security Authorization Facility (SAF) is available, the system performs SAF authorization checks on all IXGCONN REQUEST=CONNECT requests in order to protect the integrity of data in a log stream.

To connect successfully to a log stream, the caller must have SAF authorization that matches the authorization required for the log stream:
- To connect to a log stream with an authorization level of READ, the caller must have read access to RESOURCE(log_stream_name) in SAF class CLASS(LOGSTRM).
- To connect to a log stream with an authorization level of WRITE, the caller must have alter access to RESOURCE(log_stream_name) in SAF class CLASS(LOGSTRM).
If SAF is not available or if CLASS(LOGSTRM) is not defined to SAF, no security checking is performed. In that case, the caller is connected to the log stream with the requested or default AUTH parameter value.

- There is more than one version of this macro available. The parameters you can use depend on the version you specify on the PLISTVER parameter. See the description of the PLISTVER parameter for more information.

**Input register information**
Before issuing the IXGCONN macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

**Output register information**
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if register 15 contains a non-zero return code</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**
Some messages and WTORs can be issued to delay or fail the IXGCONN Request. These messages and WTORs are issued when Logger is waiting for other system services. The following messages may need to be replied to, or other action taken:

- IXG054A - LOGR CDS not yet made available for Logger's use
- IXG254I - SMS is not yet active
- IXG115A - Log stream recovery not making progress trying to move recovered log data to secondary (offload) data sets.

See the topic on IXG Messages in [z/OS MVS System Messages, Vol 10 (IXC-IZP)](https://www.ibm.com) for more information about IXG messages.
**IXGCONN macro**

**Syntax**

The standard form of the IXGCONN macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>△</td>
<td>One or more blanks must precede IXGCONN.</td>
</tr>
<tr>
<td>IXGCONN</td>
<td></td>
</tr>
<tr>
<td>△</td>
<td>One or more blanks must follow IXGCONN.</td>
</tr>
<tr>
<td>REQUEST=CONNECT</td>
<td>All parameters are valid.</td>
</tr>
<tr>
<td>REQUEST=DISCONNECT</td>
<td>STREAMTOKEN, ANSAREA, ANSLEN, USERDATA, RETCODE, RSNCODE, MF</td>
</tr>
<tr>
<td>,STREAMNAME=streamname</td>
<td>streamname: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,STREAMTOKEN=streamtoken</td>
<td>streamtoken: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ANSAREA=ansarea</td>
<td>ansarea: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ANSLEN=anslen</td>
<td>anslen: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,AUTH=READ</td>
<td>Default: AUTH=READ</td>
</tr>
<tr>
<td>,AUTH=WRITE</td>
<td></td>
</tr>
<tr>
<td>,STRUCTNAME=structname</td>
<td>structname: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,AVGBUFSIZE=avgbufsize</td>
<td>avgbufsize: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MAXBUFSIZE=maxbufsize</td>
<td>maxbufsize: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ELEMENTSIZE=elementsize</td>
<td>elementsize: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,LSVERSION=lsversion</td>
<td>lsversion: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,COMPLETEEXIT=completeexit</td>
<td>completeexit: RS-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
## IXGCONN Macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>,USERDATA=userdata</code></td>
<td><code>userdata</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,IMPORTCONNECT=NO</code></td>
<td><strong>Default</strong>: IMPORTCONNECT=NO</td>
</tr>
<tr>
<td><code>,IMPORTCONNECT=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>,DIAG=NO_DIAG</code></td>
<td><strong>Default</strong>: DIAG=NO_DIAG</td>
</tr>
<tr>
<td><code>,DIAG=NO</code></td>
<td></td>
</tr>
<tr>
<td><code>,DIAG=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>,RMNAME=rmname</code></td>
<td><code>rmname</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,RMEXIT=rmexit</code></td>
<td><code>rmexit</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,RMDATA=rmdata</code></td>
<td><code>rmdata</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,RMEVENTS=LBWRITE</code></td>
<td></td>
</tr>
<tr>
<td><code>,RMEVENTS=LBDELETE</code></td>
<td></td>
</tr>
<tr>
<td><code>,PLISTVER=IMPLIED_VERSION</code></td>
<td><strong>Default</strong>: IMPLIED_VERSION</td>
</tr>
<tr>
<td><code>,PLISTVER=MAX</code></td>
<td></td>
</tr>
<tr>
<td><code>,PLISTVER=1</code></td>
<td></td>
</tr>
<tr>
<td><code>,PLISTVER=2</code></td>
<td></td>
</tr>
<tr>
<td><code>,RETCODE=retcode</code></td>
<td><code>retcode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,RSNCODE=rsncode</code></td>
<td><code>rsncode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>,MF=S</code></td>
<td><strong>Default</strong>: MF=S</td>
</tr>
<tr>
<td><code>,MF=(L,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>,MF=(L,list addr,attr)</code></td>
<td></td>
</tr>
<tr>
<td><code>,MF=(L,list addr,0D)</code></td>
<td></td>
</tr>
<tr>
<td><code>,MF=(E,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>,MF=(E,list addr,COMPLETE)</code></td>
<td></td>
</tr>
<tr>
<td><code>,MF=(E,list addr,NOCHECK)</code></td>
<td></td>
</tr>
<tr>
<td><code>,MF=(M,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>,MF=(M,list addr,COMPLETE)</code></td>
<td></td>
</tr>
<tr>
<td><code>,MF=(M,list addr,NOCHECK)</code></td>
<td></td>
</tr>
</tbody>
</table>
IXGCONN macro

Parameters

The parameters are explained as follows:

REQUEST=CONNECT
REQUEST=DISCONNECT

Input parameter specifying whether the program is connecting to or disconnecting from the specified log stream.

When you specify CONNECT, all parameters are valid. Keywords required with connect are: STREAMNAME, STREAMTOKEN, ANSAREA, and ANSLEN.

When you specify DISCONNECT, the following parameters are valid (required parameters are underlined): STREAMTOKEN, ANSAREA, ANSLEN, USERDATA, RETCODE, RSNCODE, and MF.

,STREAMNAME=streamname
Specifies the 26-byte field (or register) containing the name of the log stream to which a program is connecting. You must use the name you defined for the log stream in the LOGR policy, see the IXGINVNT macro for information on the syntax of log stream names in the LOGR policy.

,STREAMTOKEN=streamtoken
Specifies the 16-byte token uniquely identifying the program’s connection to the log stream.

When specified with REQUEST=CONNECT, STREAMTOKEN is an output parameter where IXGCONN places the log stream token when the macro completes successfully.

When specified with REQUEST=DISCONNECT or other logger services, STREAMTOKEN is an input parameter where you specify the log stream token returned at connection.

,ANSAREA=ansarea
Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

,ANSLEN=anslen
Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

,AUTH=READ
,AUTH=WRITE

Specifies whether the caller has write or read access to the specified log stream.

If you specify AUTH=READ when connecting to a log stream, the program must also have read access authority to SAF resource(logstream_name) in CLASS(LOGSTRM) for the specified log stream. You can then issue only the IXGBRWSE and IXGQUERY requests against the log stream.

If you specify AUTH=WRITE when connecting to a log stream, the program must also have write access authority to SAF resource(logstream_name) in CLASS(LOGSTRM) for the specified log stream. You can then issue any system logger request against the log stream.


**IXGCONN macro**

`,STRUCTNAME=structname`

Specifies the name or address (using a register) of a 16-byte output field where IXGCONN REQUEST=CONNECT will return the name of the coupling facility structure that the log stream is connected to. The name comes from the LOGR policy.

If you are connecting to a DASD-only log stream, this field will contain binary zeros. In addition, flag Ansaa_DasdOnlyLogStream in macro IXGANSAA will be set on for a DASD-only log stream.

`,MAXBUFSIZE=maxbufsize`

Specifies the name or address (using a register) of a 4-byte output field where IXGCONN returns the size, in bytes, of the largest log block that can be written to this log stream.

MAXBUFSIZE is defined in the LOGR policy.

`,AVGBUFSIZE=avgbufsize`

Specifies the name or address (using a register) of a 4-byte output field where IXGCONN returns the average size, in bytes, of individual log blocks that can be written to the coupling facility structure associated with this log stream.

AVGBUFSIZE is defined in the LOGR policy.

- If you are using a LOGR couple data set for a coupling facility log stream, this value shows the initial setting used to determine the element-to-entry ratio. System logger monitors structure usage and adjusts the average buffer size dynamically, but the AVGBUFSIZE value returned by IXGCONN will always reflect the original setting rather than the actual value in use by system logger at any given time.

- If you are connecting to a DASD-only log stream, this field will contain binary zeros. In addition, flag Ansaa_DasdOnlyLogStream in macro IXGANSAA will be set on for a DASD-only log stream.

`,ELEMENTSIZE=elementsize`

Specifies the name or address (using a register) of a 4-byte output field where IXGCONN returns the size of the elements that system logger will break the log blocks into to write them to the coupling facility associated with this log stream.

If you are connecting to a DASD-only log stream, this field will contain binary zeros. In addition, flag Ansaa_DasdOnlyLogStream in macro IXGANSAA will be set on for a DASD-only log stream.

`,LSVERSION=lsversion`

Specifies the name or address (using a register) of a 64-bit output field where IXGCONN returns the version of the log stream the program is connecting to.

The log stream version is a UTC timestamp that uniquely identifies the instance of the log stream definition. A program can use the log stream version to see if a log stream definition has been deleted and redefined since the last connect to a log stream.

For example, assume you connect to log stream LS1 and IXGCONN returns a log stream version of X‘AA00000000000000’, which the program saves. On a subsequent connection to log stream LS1, IXGCONN returns a different log stream version, which indicates that the definition for log stream LS1 in the LOGR policy has been deleted and redefined since the last connection.

`,COMPLETEEXIT=completeexit`

Specifies the name or address (using a register) of a user exit called a complete exit. Use this parameter to specify a complete exit for the caller. The complete
exit gets control when the system processes IXGBRWSE, IXGDELET, or IXGWRITE requests that specify MODE=SYNCEXIT asynchronously. The complete exit receives control in SRB mode, supervisor state, key 0, enabled, and unlocked.

If you specify a name for this parameter, it must be the name of an entry point addressable in the invoking load module. For example, the name can be a routine name that exits in the invoking module or a CSECT link-edited into the invoking load module.

The caller must ensure that the complete exit routine is loaded into either private storage in the connector's primary address space or common storage. The exit must remain loaded in storage until all asynchronous requests that have specified SYNCEXIT with the log stream token returned by this connect request have completed. Even if the log stream is disconnected, you cannot assume that all SYNCEXIT requests have completed.

For more information on coding a complete exit and the environment where the complete exit runs, see [z/OS MVS Programming: Assembler Services Guide](https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/cics help/en/cics help_220.htm).

**USERDATA=userdata**

Specifies a 64-byte input/output field containing a user data area.

When specified with REQUEST=CONNECT, USERDATA is an output parameter where IXGCONN returns the user data specified for this log stream.

When specified with REQUEST=DISCONNECT, USERDATA is an input parameter where you can specify or update the user data the user data for the specified log stream. You can only specify or change the user data for a log stream on a disconnect request.

**IMPORTCONNECT=NO**

**IMPORTCONNECT=YES**

Specifies whether the connection is for writing or importing log data to a log stream. You must specify AUTH=WRITE to use the IMPORTCONNECT parameter.

If you specify IMPORTCONNECT=YES, this connection will be used for importing data to a log stream. Importing log data means using the IXGIMPRT service to copy data from one log stream to another, maintaining the same log block identifier and UTC time stamp. IXGWRITE requests are not valid with IMPORTCONNECT=YES. You can have only one IMPORTCONNECT=YES connection active for a log stream in the sysplex.

If you specify IMPORTCONNECT=NO, which is the default, the connect request is a write connection. In a write connection, only IXGWRITE requests can be issued against the log stream, IXGIMPRT requests will be rejected.

You can have multiple write connects to a log stream, provided there are no import connections. If you have a write connect established against a log stream, a subsequent import connection will be rejected. You cannot, in other words, issue both IXGIMPRT and IXGWRITE requests against a single log stream.

**RMNAME=rmname**

Specifies the name (or address in a register) of the 8-byte input field containing the name of the resource manager program connecting to the log stream. The resource manager name specified on the IXGCONN request must be the same as the one associated with the log stream in the log stream definition in the LOGR policy. The application must run in supervisor state, key 0-7 to use this parameter.
The RMNAME parameter is specified only by the resource manager at connect time, to tell system logger that it is connecting to a log stream. Other connections to a resource manager managed log stream do not have to specify RMNAME. Note that a resource manager can only connect to one log stream per system.

The active primary LOGR couple data set must be formatted at the z/OS level to use this parameter.

```
,RMEXIT=rmexit
```

 specifies the name (or address in a register) of the input field containing the address of user exit for the resource manager. The application must run in supervisor state, key 0-7 to use this parameter.

RMEXIT is required with the RMNAME parameter, even though use of a resource manager exit is optional. The exit is called only if the resource manager monitors write and/or delete events as selected on the RMEVENT parameter.

The active primary LOGR couple data set must be formatted at the z/OS level to use this parameter.

RMEXIT is required when you specify RMNAME.

```
,RMDATA=rmdata
```

 specifies the name (or address in a register) of the 8-byte input field containing the data for the user exit. The application must run in supervisor state, key 0-7 to use this parameter.

RMDATA is required with the RMNAME parameter.

The active primary LOGR couple data set must be formatted at the z/OS level to use this parameter.

```
,RMEVENT=LBWRITE
,RMEVENT=LBDELETE
```

 Input parameter specifying the events that you want to trigger the resource manager user exit. RMEVENT is required with the RMNAME parameter. You can specify RMEVENTS=LBWRITE, RMEVENTS=LBDELETE, or RMEVENTS=(LBWRITE, LBDELETE). The application must run in supervisor state, key 0-7 to use this parameter.

If you specify RMEVENT=LBWRITE, successful write requests to the log stream will trigger the resource manager user exit.

If you specify RMEVENT=LBDELETE, successful delete requests to the log stream will trigger the resource manager user exit.

The active primary LOGR couple data set must be formatted at the z/OS level to use this parameter.

```
,DIAG=NO_DIAG
,DIAG=NO
,DIAG=YES
```

 Specifies whether Logger should provide additional diagnostics as specified on the logstream definition DIAG parameter. This indication is used over the span of this connection. Refer to the DIAG keyword on the IXGINVNT, IXGBRWSE, and IXGDELET macro services.

If you specify DIAG=NO_DIAG, which is the default, then Logger will not provide the additional diagnostics as specified on the logstream definition DIAG parameter, unless another Logger service, for example, IXGBRWSE, specifically requests the additional diagnostics.
IXGCONN macro

If you specify DIAG=NO, the Logger will not provide the additional diagnostics as specified on the logstream definition DIAG parameter, regardless of other Logger service specifications.

If you specify DIAG=YES, then Logger will provide additional diagnostics as specified on the logstream definition DIAG parameter, unless another Logger service, for example, IXGDELETE, specifically requests not to provide the additional diagnostics.

,PLISTVER=IMPLIED VERSION, PLISTVER=MAX, PLISTVER=1, PLISTVER=2

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- **IMPLIED VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **1**, which supports all parameters except those specifically referenced in higher versions.

- **2**, which supports both the following parameters and parameters from version 1:
  - COMPLETEEXIT
  - IMPORTCONNECT
  - LSVERSION
  - RMDATA
  - RMEVENTS
  - RMEXIT
  - RMNAME

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1 or 2

,RETCODE=retcode

Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=rsncode

Specifies a name (or address in a register) of a 4-byte output field where the
system will place the reason code. The reason code is also in general purpose
register (GPR) 0, if you received a non-zero return code.

\[MF=S\]
\[MF=(L,list \ addr)\]
\[MF=(L,list \ addr,attr)\]
\[MF=(L,list \ addr,0D)\]
\[MF=(E,list \ addr)\]
\[MF=(E,list \ addr,COMPLETE)\]
\[MF=(E,list \ addr,NOCHECK)\]
\[MF=(M,list \ addr)\]
\[MF=(M,list \ addr,COMPLETE)\]
\[MF=(M,list \ addr,NOCHECK)\]

Use MF=S to specify the standard form of the macro, which builds an inline
parameter list and generates the macro invocation to transfer control to the
service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with
the execute form of the macro for applications that require reentrant code. The
list form defines an area of storage that the execute form uses to store the
parameters. Only the PLISTVER parameter can be specified on the list form of
the macro. IBM recommends that you always specify PLISTVER=MAX on the
list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form
together with the list form of the macro for applications that require reentrant
code. The execute form of the macro stores the parameters into the storage area
defined by the list form, and generates the macro invocation to transfer control
to the service.

Use MF=M together with the list and execute forms of the macro for service
routines that need to provide different options according to user-provided
input. Use the list form to define a storage area; use the modify form to set the
appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following
order:

- Use MF=(M,list \ addr,COMPLETE), specifying appropriate parameters,
  including all required ones.
- Use MF=(M,list \ addr,NOCHECK), specifying the parameters you want to
  change.
- Use MF=(E,list \ addr,NOCHECK), to execute the macro.

\,list \ addr
The name of a storage area to contain the parameters.

\,attr
An optional 1- to 60-character input string, which can contain any value
that is valid on an assembler DS pseudo-op. You can use this parameter to
force boundary alignment of the parameter list. If you do not code \attr, the
system provides a value of 0D, which forces the parameter list to a
doubleword boundary.

\,COMPLETE
Specifies that the system is to check for required parameters and supply
defaults for omitted optional parameters.

\,NOCHECK
Specifies that the system is not to check for required parameters and is not
to supply defaults for omitted optional parameters.
ABEND codes

None.

Return and reason codes

When IXGCONN macro returns control to your program, GPR 15 contains a return code and GPR 0 contains a reason code.

The IXGCON mapping macro provides equate symbols for the return and reason codes. The equate symbols associated with each hexadecimal return code are as follows:

- 00 IXGRETCODEOK - Service completes successfully.
- 04 IXGRETCODEWARNING - Service completes with a warning.
- 08 IXGRETCODEERROR - Service does not complete.
- 0C IXGRETCODECOMPERROR - Service does not complete.

The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>xxxx0000</td>
<td>Equate Symbol: IxgRsnCodeOk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Explanation: Request processed successfully.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0404</td>
<td>Equate Symbol: IxgRsnCodeDisconnectInProgress</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Explanation: Environment error. The disconnect request is being completed asynchronously. The application has been disconnected from the log stream and the stream token is no longer valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: The log stream cannot be deleted until the asynchronous portion of the disconnect processing completes.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0406</td>
<td>Equate Symbol: IxgRsnCodeConnectRebuild</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Explanation: Environment error. The connect request was successful, but the log stream is temporarily unavailable because a coupling facility structure re-build is in progress.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Listen to the ENF signal 48, which will indicate either that the log stream is available because the re-build completed successfully or that the log stream is not available because the re-build failed. In the meantime, do not attempt to issue system logger services against the log stream.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Reason Code</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| 04          | xxx0407     | **Equate Symbol:** IxgRsnCodeConnPossibleLossOfData  
**Explanation:** Environment error. The request was successful, but there may be log blocks permanently missing between this log block and the one previously returned. This condition occurs when a system or coupling facility fails and not all of the data in the log stream could be recovered.  
**Action:** If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss. |
| 04          | xxx0408     | **Equate Symbol:** IxgRsnCodeDsDirectoryFullWarning  
**Explanation:** Environment error. The request was successful, but the DASD data set directory for the log stream is now full. system logger cannot offload any further data to DASD. system logger will continue to process IXGWRITE requests only until the coupling facility structure space for this log stream is full.  
**Action:** Either delete data from the log stream to free up space in the data set directory or disconnect from the log stream. |
| 04          | xxx0409     | **Equate Symbol:** IxgRsnCodeWowWarning  
**Explanation:** Environment error. The request was successful, but an error condition was detected during a previous offload of data. system logger might not be able to offload further data. system logger will continue to process IXGWRITE requests only until the interim storage for the log stream is filled. (Interim storage is the coupling facility for a coupling facility log stream and local storage buffers for a DASD-only log stream.)  
**Action:** Do not issue any further requests for this log stream and disconnect. Connect to another log stream. Check the system log for message IXG301I to determine the cause of the error. If you cannot fix the error, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
| 08          | xxx0801     | **Equate Symbol:** IxgRsnCodeBadParmlist  
**Explanation:** Program error. The parameter list could not be accessed.  
**Action:** Ensure that the storage area for the parameter list is accessible to the system logger for the duration of the request. The parameter list storage must be addressable in the caller’s primary address space and in the same key as the caller. |
Table 118. Return and Reason Codes for the IXGCONN Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0802    | **Equate Symbol:** IxgRsnCodeXESError  
**Explanation:** System error. A severe cross-system extended services (XES) error has occurred.  
**Action:** See ANSAA_DIAG1 for the XES return code and ANSAA_DIAG2 for the XES reason code. |
| 08          | xxxx0806    | **Equate Symbol:** IxgRsnCodeBadStmToken  
**Explanation:** Program error. The stream token was not valid.  
**Action:** Make sure that the stream token specified is valid. |
| 08          | xxxx0808    | **Equate Symbol:** IxgRsnCodeEIOError  
**Explanation:** System error. A severe log data set I/O error has occurred.  
**Action:** Contact the IBM Support Center. Provide the return and reason code. |
| 08          | xxxx080A    | **Equate Symbol:** IxgRsnCodeRequestLocked  
**Explanation:** Program error. The program issuing the request is holding a lock.  
**Action:** Ensure that the program issuing the request is not holding a lock. |
| 08          | xxxx080B    | **Equate Symbol:** IxgRsnCodeNoStream  
**Explanation:** Program error. The log stream name specified has not been defined in the LOGR policy.  
**Action:** Ensure that the required log stream name has been defined in the LOGR policy. If the definition appears to be correct, ensure that the application is passing the correct log stream name to the service. |
| 08          | xxxx080C    | **Equate Symbol:** IxgRsnCodeStagingAllocError  
**Explanation:** Environment error. The system encountered a severe dynamic allocation error with the staging data set. ANSAA_DIAG2 of the answer area contains either the dynamic allocation error code, SMS reason code, or media manager reason code. For more information about the error, check for either message IXG251I, which is issued for data set allocation errors, or check for messages issued by the access method.  
**Action:** If the problem persists, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
### IXGCONN macro

#### Table 118. Return and Reason Codes for the IXGCONN Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx080D     | **Equate Symbol:** IxgRsnCodeNoSAFAuth  
**Explanation:** Environment error. The user does not have correct SAF authorization for the request. The caller is not authorized to connect to the log stream or the caller specified AUTH=WRITE when connecting to a log stream with only READ authority.  
**Action:** IXGCONN returns information about the error in the answer area that is mapped by IXGANSAA. Investigate the meaning of ANSAA_Diag1, ANSAA_Diag2 and ANSAA_Diag4.  
• ANSAA_Diag1 contains the RACF or installation exit return code from the RACROUTE REQUEST=AUTH macro.  
• ANSAA_Diag2 contains the RACF or installation exit reason code from the RACROUTE REQUEST=AUTH macro.  
• ANSAA_Diag4 contains the SAF return code from the RACROUTE REQUEST=AUTH macro.  
See [z/OS Security Server RACROUTE Macro Reference](https://www.ibm.com/support/docview.wss?uid=swg27046606) for information about the RACROUTE macro.  
Define the required SAF authorization to allow the requestor to connect to the log stream. If authorization has already been defined, either change the authorization to allow UPDATE access to the log stream or change the application to AUTH=READ. |
| 08          | xxxx0811     | **Equate Symbol:** IxgRsnCodeBadStrname  
**Explanation:** Environment error. The structure name specified on the STRUCTNAME parameter is not defined in the CFRM policy.  
**Action:** Make sure that the structure you want to specify is defined in the CFRM policy. |
| 08          | xxxx0812     | **Equate Symbol:** IxgRsnCodeLogStreamRecoveryFailed  
**Explanation:** Environment error. The log stream could not be recovered so the connection attempt failed. The system issues message IXG210E and/or IXG211E along with message IXG231I providing further information about the error.  
**Action:** If the problem persists, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. |
### IXGCONN macro

**Table 118. Return and Reason Codes for the IXGCONN Macro  (continued)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0813    | **Equate Symbol**: IxgRsnCodeLogStreamDeleted  
**Explanation**: Environment error. The request to connect to the specified log stream failed because the log stream is being deleted.  
**Action**: Re-define the log stream in the LOGR policy and then re-issue the connect request. |
| 08          | xxxx0814    | **Equate Symbol**: IxgRsnCodeNotAvailForIPL  
**Explanation**: Environment error. The system logger address space is not available for the remainder of this IPL. The system issues messages about this error during system logger initialization.  
**Action**: See the explanation for system messages issued during system logger initialization. |
| 08          | xxxx0815    | **Equate Symbol**: IxgRsnCodeNotEnabled  
**Explanation**: Program error. The program issuing the request is not enabled for I/O and external interrupts, so the request fails.  
**Action**: Make sure the program issuing the request is enabled for I/O and external interrupts. |
| 08          | xxxx0816    | **Equate Symbol**: IxgRsnCodeBadAnslen  
**Explanation**: Program error. The answer area length (ANSLEN parameter) is not large enough. The system logger returned the required size in the Ansaa_Preferred_Size field of the answer area, mapped by IXGANSSAA macro.  
**Action**: Re-issue the request, specifying an answer area of the required size. |
| 08          | xxxx0819    | **Equate Symbol**: IxgRsnCodeSRBMode  
**Explanation**: Program error. The calling program is in SRB mode, but task mode is the required dispatchable unit mode for this system logger service.  
**Action**: Make sure the calling program is in task mode. |
<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>xxxx081A</td>
<td><strong>Equate Symbol</strong>: IxgRsnCodeMaxStreamConn &amp; IXGINVNT requests</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation</strong>: Environment error. This system has reached the limit for the maximum number of log streams that can be concurrently active. One of the following is true:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The limit of 16,384 concurrently active DASDONLY log streams per system has been reached. For this case, the Answer Area field DIAG1 will contain 16,384.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Either the PRODUCTION or TEST GROUP cannot connect to any more log streams. Message IXG075E or IXG076I is issued. In this case, the Answer Area field DIAG1 will contain the number of structures that are in use for this GROUP.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The TEST GROUP has previously failed and a request has been made to define a logstream with GROUP(TEST). Message IXG074I has been previously issued. In this case, the Answer Area field DIAG1 will contain 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A Log stream delete cannot be processed because logger needs to perform an internal connect to the Log stream to complete the delete but no more connections are allowed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Your workload need to be planned to either consolidate log streams or balance system activity such that fewer log streams are needed during this time frame.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx081B</td>
<td><strong>Equate Symbol</strong>: IxgRsnCodePrimaryNotHome</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation</strong>: Program error. The primary address space does not equal the home address space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Make sure that the primary address space equals the home address space when issuing this system logger service.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx081D</td>
<td><strong>Equate Symbol</strong>: IxgRsnCodeRMNameBadState</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation</strong>: Program error. The calling program cannot issue IXGCONN with the RMNAME parameter unless it is in supervisor state and system key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Make sure the calling program is in supervisor state.</td>
</tr>
<tr>
<td>08</td>
<td>xxxx081E</td>
<td><strong>Equate Symbol</strong>: IxgRsnCodeXESStrNotAuth</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation</strong>: Environment Error. The system logger address space does not have access authority to the coupling facility structure associated with the log stream specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Make sure the system logger address space has SAF access to the structure.</td>
</tr>
</tbody>
</table>
### IXGCONN macro

#### Table 118. Return and Reason Codes for the IXGCONN Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08xxxx081F |             | **Equate Symbol:** IxgRsnCodeXcdsError  
**Explanation:** System error. System logger encountered an internal problem while processing the LOGR couple data set.  
**Action:** Contact the IBM Support Center. Provide the return and reason code and the contents of the answer area (ANSAREA field). |
| 08xxxx0820 |             | **Equate Symbol:** IxgRsnCodeBadModelConn  
**Explanation:** Program error. The program issued an IXGCONN request to connect to a log stream that was defined as a model in the LOGR policy. You cannot connect to a model log stream.  
**Action:** Either change the definition of the specified structure so that it is not a model, or else request connection to a different log stream that is not a model. |
| 08xxxx082D |             | **Equate Symbol:** IxgRsnCodeExpiredStmToken  
**Explanation:** Environment error. The stream token is no longer valid because the connector has been disconnected.  
**Action:** Connect to the log stream again before issuing any functional requests. |
| 08xxxx082E |             | **Equate Symbol:** IxgRsnCodeNoLogrCDSAvail  
**Explanation:** Environment error. The request failed because no LOGR couple data set is available. The operator was prompted to either make a couple data set available or to indicate that the current request should be rejected. The operator specified that the current request should be rejected.  
**Action:** System logger services are unavailable for the remainder of this IPL. |
| 08xxxx0831 |             | **Equate Symbol:** IxgRsnCodeBadStreamName  
**Explanation:** Program error. The log stream name specified on the STREAMNAME parameter is not valid.  
**Action:** Issue the request again with a valid log stream name on the STREAMNAME parameter. |
| 08xxxx083A |             | **Equate Symbol:** IxgRsnCodeRMNameNotAllowed  
**Explanation:** Program error. The request specified the RMNAME parameter, but the log stream is not defined as having an associated resource manager.  
**Action:** Either define a resource manager for the log stream definition in the LOGR couple data set, or remove the RMNAME parameter from the request. |
### Table 118. Return and Reason Codes for the IXGCONN Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0843    | **Equate Symbol:** IxgRsnCodeXcdsReformat  
**Explanation:** Program error. A couple data set record is not valid.  
**Action:** Format the system logger couple data set again. |
| 08          | xxxx084C    | **Equate Symbol:** IxgRsnCodeRMAlreadyConnected  
**Explanation:** Program error. The resource manager is trying to connect to a log stream that it is already connected to. Only one connection specifying RMNAME can be active for a log stream.  
**Action:** Correct the program so that it does not try to reconnect to the log stream. |
| 08          | xxxx084F    | **Equate Symbol:** IxgRsnCodeInvalidRMNameSpecified  
**Explanation:** Program error. The value for the RMNAME parameter on the connect request does not match the name of the resource manager defined in the LOGR couple data set for the log stream.  
**Action:** Either correct the RMNAME value on the connect request or correct the resource manager name in the log stream definition in the LOGR couple data set. |
| 08          | xxxx0850    | **Equate Symbol:** IXGRSNCODEBADVECTORLEN  
**Explanation:** Environment error. The connect request was rejected. System logger was unable to locate a vector table in the hardware system area (HSA) that is large enough for the number of log streams associated with it.  
**Action:** Add storage to the vector storage table and/or retry the connect request later, when storage might be available. |
| 08          | xxxx0851    | **Equate Symbol:** IXGRSNCODEBADCFLEVEL  
**Explanation:** Environment error. The connect request was rejected. The operational level of the coupling facility is not sufficient to support logger functions.  
**Action:** Ensure that the coupling facility operational level for logger structures is at the required level. See<sup>1</sup>[z/OS MVS Setting Up a Sysplex](https://www.ibm.com)
### IXGCONN macro

Table 118. Return and Reason Codes for the IXGCONN Macro  (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0853    | **Equate Symbol:** IXgRsnCodeNoCF  
**Explanation:** Environment error. The connect request was rejected. System logger could not allocate coupling facility structure space because no suitable coupling facility was available.  
**Action:** Check accompanying message IXG206I for a list of the coupling facilities where space allocation was attempted and the reason why each attempt failed. |
| 08          | xxxx0861    | **Equate Symbol:** IXgRsnCodeRebuildInProgress  
**Explanation:** Environment error. No requests can be processed for this log stream because a coupling facility structure re-build is in progress for the structure associated with this log stream.  
**Action:** Listen for ENF signal 48 that will indicate one of the following:  
* The log stream is available because the re-build completed successfully. Re-issue the request.  
* The re-build failed and the log stream is not available. |
| 08          | xxxx0862    | **Equate Symbol:** IXgRsnCodeXESPurge  
**Explanation:** Environment error. An cross-system extended services (XES) request has been purged due to re-build processing.  
**Action:** Listen for ENF signal 48 that will indicate one of the following:  
* The log stream is available because the re-build completed successfully. Re-issue the request.  
* The re-build failed and the log stream is not available. |
| 08          | xxxx0863    | **Equate Symbol:** IXgRsnCodeSTRUCTUREFAILED  
**Explanation:** Environment error. Either the coupling facility structure associated with the log stream has failed or the coupling facility itself has failed.  
**Action:** Listen for ENF signal 48 that will indicate one of the following:  
* The log stream is available because the re-build completed successfully. Re-issue the request.  
* The re-build failed and the log stream is not available. |
<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0866       | **Equate Symbol:** IXGRSNCODENOCONNECTIVITY  
**Explanation:** Environment error. No connectivity exists to the coupling facility associated with the log stream. The system logger will either attempt to re-build the log stream in another coupling facility or the log stream will be disconnected.  
**Action:** Listen for ENF signal 48 that will indicate one of the following:  
- The log stream is available because the re-build completed successfully. Re-issue the request.  
- The re-build failed and the log stream is not available.  
- The log stream has been disconnected from this system.  
If a re-build initiated because of a loss of connectivity previously failed, an ENF corresponding to this reason code might not be issued. Further action by the installation might be necessary to cause the change of the log stream status again. Check the log for messages IXG101I, IXG107I and related rebuild messages for information on resolving any outstanding issues. |
| 08          | xxxx0866       | **Equate Symbol:** IXGRSNCODESTRUCTUREFULL  
**Explanation:** Environment error. The coupling facility structure space is full.  
**Action:** Listen to the ENF signal 48 which will indicate that space is available for the structure after data has been offloaded to DASD. |
| 08          | xxxx0890       | **Equate Symbol:** IXGRSNCODEADDRSPACENOTAVAIL  
**Explanation:** System error. The system logger address space failed and is not available.  
**Action:** Do not issue system logger requests. |
| 08          | xxxx0891       | **Equate Symbol:** IXGRSNCODEADDRSPACEINITIALIZING  
**Explanation:** System error. The system logger address space is not available because it is IPLing.  
**Action:** Listen for ENF signal 48, which will indicate when the system logger address space is available. Re-issue this request. You can also listen for ENF signal 48, which will indicate if the system logger address space will not be available for the life of the IPL. In that case, do not issue system logger services.
<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx08B0     | **Equate Symbol:** IXGRSNCODESTRUCTURENOTAVAIL  
**Explanation:** Environment error. The connect request failed. The structure associated with the log stream is temporarily unavailable because either a re-build is in progress, a structure dump is in progress, or connections to the structure are being prevented.  
**Action:** Listen for ENF signal 48, which indicates that a coupling facility is available, and then retry the connect. |
| 08          | xxxx08D3     | **Equate Symbol:** IXGRsnCodeFuncNotSupported  
**Explanation:** Environment error. The connect request specified the RMNAME or IMPORTCONNECT parameter. The request failed because the active primary LOGR couple data set must be at the z/OS level to support these parameters.  
**Action:** Either retry the request without the RMNAME or IMPORTCONNECT parameters or reformat the LOGR couple data set to the z/OS level. |
| 08          | xxxx08D6     | **Equate Symbol:** IXGRsnCodeConnTypeNotAllowed  
**Explanation:** Environment error. One of the following occurred:
- The connect request specified IMPORTCONNECT=YES, but there is already an active write connection (AUTH=WRITE IMPORTCONNECT=NO) in the sysplex. You cannot have an import connection and a write connection to the same log stream.  
- The connect request specified AUTH=WRITE IMPORTCONNECT=NO, but there is already an active import connection (IMPORTCONNECT=YES) for the log stream. You cannot have an import connection and a write connection to the same log stream.  
You can only have one import connection to a log stream. You may have multiple write connections, as long as there is no import connection against a log stream.  
**Action:** Correct your program and retry the request. |
Table 118. Return and Reason Codes for the IXGCONN Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx08E2    | **Equate Symbol:** IxgRsnCodeDasdOnlyConnected  
**Explanation:** Environment error system logger rejected an attempt to connect to a DASD-only log stream because the log stream is already connected to by another log stream in the sysplex. Only one system at a time can connect to a DASD-only log stream.  
**Action:** Determine which system you want to have a connection to the log stream. If you need this connection, disconnect the first system connection to the log stream and retry this connect request. |
| 08          | 000008E3    | **Equate Symbol:** IxgRsnCodeLogstreamNotSupported  
**Explanation:** Environment error. An attempt to connect for the log stream is rejected on this system because the system release level does not support this type of log stream. For example, this system does not support DASD-only log streams, or a log stream attribute such as EHLQ or DUPLEXMODE(DRXRC) cannot be processed on this system release level.  
**Action:** If you must connect to a DASD-only log stream, make sure you do one of the following:  
- Update the log stream definition in the LOGR policy to a coupling facility one by specifying a structure name on the definition.  
- To issue a request for a log stream that has the EHLQ attribute, you must be on a system that is at z/OS Version 1 Release 3 or higher.  
If you must connect to a log stream with the EHLQ attribute specified, make sure you connect from a system that is at z/OS Version 1 Release3 or higher.  
If you must connect to a log stream with the DUPLEXMODE(DRXRC) attribute specified, make sure you connect from a system that is at z/OS Version 1 Release 7 or higher. |
| 0C          | xxxx0000    | **Equate Symbol:** IxgRetCodeCompError  
**Explanation:** User or System error. One of the following occurred:  
- You issued the FORCE IXGLOGR,ARM command to terminate the system logger address space.  
- system logger component error occurred.  
**Action:** If this reason code is not the result of forcing the system logger address space, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. Provide the diagnostic data in the answer area (IXGANSAA) and any dumps or LOGREC entries from system logger. |
Example 1

Issue IXGCONN REQUEST=CONNECT to connect to a log stream with write authority.

```
IXGCONN REQUEST=CONNECT,
STREAMNAME=STRMNAME,
STREAMTOKEN=XTOKEN,
AUTH=WRITE,
ANSAREA=ANSAREA,
ANSLEN=ANSLEN,
RSNCODE=RSNCODE,
MF=S,
RETCODE=RETCODE
```

```
STRMNAME DC CL26'LOG.STREAM.NAME' stream name
ANSLEN DC A(L'ANSAREA) length of logger's answer area
TOKEN DS CL16 returned stream token
ANSAREA DS CL(ANSAA_LEN) answer area for log requests
RETCODE DS F return code from logger
RSNCODE DS F reason code from logger
DATAREA DSECT
IXGANSAA LIST=YES answer area
```

Example 2

Issue IXGCONN REQUEST=CONNECT using registers.

```
LA R6,STRMNAME load stream name into reg 6
IXGCONN REQUEST=CONNECT,
STREAMNAME=(6),
STREAMTOKEN=XTOKEN,
AUTH=WRITE,
ANSAREA=ANSAREA,
ANSLEN=ANSLEN,
RSNCODE=RSNCODE,
MF=S,
RETCODE=RETCODE
```

```
STRMNAME DC CL26'LOG.STREAM.NAME' stream name
ANSLEN DC A(L'ANSAREA) length of logger's answer area
TOKEN DS CL16 returned stream token
ANSAREA DS CL(ANSAA_LEN) answer area for log requests
RETCODE DS F return code from logger
RSNCODE DS F reason code from logger
DATAREA DSECT
IXGANSAA LIST=YES answer area
R6 EQU 6 set up register 6
```

Example 3

Issue IXGCONN REQUEST=CONNECT as an import connect. This means the connection may issue IXGIMPRT to import data to a log stream.

```
IXGCONN REQUEST=CONNECT,
STREAMNAME=ONAME,
STREAMTOKEN=OTOKEN,
AUTH=WRITE,
IMPORTCONNECT=YES,
ANSAREA=ANSAREA,
ANSLEN=ANSLEN,
RSNCODE=RSNCODE
```

```
ONAME DS CL26 Output Stream name
STOKEN DS CL16 Input Stream token
XANSAREA DS CL(ANSAA_LEN) Logger answer area
XANSLEN DC A(ANSAA_LEN) Answer area length
Example 4

Issue IXGCONN REQUEST=DISCONNECT to disconnect from a log stream and associate some user data with the log stream.

```assembly
   IXGCONN REQUEST=DISCONNECT, X
   STREAMTOKEN=TOKEN, X
   USERDATA=USERDATA, X
   ANSAREA=ANSAREA, X
   ANSLEN=ANSLEN, X
   RSNCODE=RSNCODE, X
   MF=S, X
   RETCODE=RETCODE
   USERDATA DC CL64'SOME USER DATA' user data to log with DISCONNECT
   ANSLEN DC A(L'ANSAREA) length of logger's answer area
   TOKEN DS CL16 token returned from CONNECT
   ANSAREA DS CL(ANSAA_LEN) answer area for log requests
   RETCODE DS F return code from logger
   RSNCODE DS F reason code from logger
   DATAREA DSECT
   IXGANSAA LIST=YES answer area
```

Example 5

Issue IXGCONN to connect to a log stream, specifying a resource manager and resource manager exit for the log stream.

```assembly
   L R5,RMEXIT_ADDR
   IXGCONN REQUEST=CONNECT,
   STREAMNAME=SNAME,
   STREAMTOKEN=STOKEN,
   AUTH=WRITE,
   RMNAME=RMNAME,
   RMEXIT=(R5),
   RMDATA=RMDATA,
   RMEVENTS=(LBWRITE,LDELETE),
   ANSAREA=ANSAREA,
   ANSLEN=ANSLEN,
   RSNCODE=RSNCODE
```

*  

```assembly
   SNAME DS CL26 Stream name
   STOKEN DS CL16 Input Stream token
   RMEXIT_ADDR DS A RM exit rtn address
   RMDATA DS CLB RM exit data
   XANSAREA DS CL(ANSAA_LEN) Logger answer area
   XANSLEN DC A(ANSAA_LEN) Answer area length
   RSNCODE DS F Reason code
   DATAREA DSECT
   IXGANSAA LIST=YES The answer area macro
```
IXGCONN macro
Chapter 133. IXGDELET — Deleting log data from a log stream

Description

Use the IXGDELET macro to delete log blocks from a log stream.

For information about using the system logger services and the system logger inventory, see [z/OS MVS Programming: Assembler Services Guide](#) which includes information about related macros IXGCONN, IXGBRWSE, IXGWRITE, IXGINVNT, and IXGQUERY.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state with any PSW key. The caller must be</td>
</tr>
<tr>
<td></td>
<td>supervisor state with any system (0-7) PSW key to either</td>
</tr>
<tr>
<td></td>
<td>invoke the service in SRB mode or use the</td>
</tr>
<tr>
<td></td>
<td>MODE=SYNCEXIT keyword.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held.</td>
</tr>
<tr>
<td>Control parameter:</td>
<td>All control parameters must be in the primary address space</td>
</tr>
<tr>
<td></td>
<td>with the following exceptions:</td>
</tr>
<tr>
<td></td>
<td>• The ECB should be addressable from the home address space.</td>
</tr>
<tr>
<td></td>
<td>• All storage areas specified must be in the same storage</td>
</tr>
<tr>
<td></td>
<td>key as the caller.</td>
</tr>
</tbody>
</table>

Programming requirements

- The current primary address space must be the same primary address space used at the time your program issued the IXGCONN request.
- The parameter list for this service must be addressable in the caller’s primary address space.
- The calling program must be connected to the log stream with write authority through the IXGCONN service.
- Include the IXGCON mapping macro in your program. This macro provides a list of equate symbols for the system logger services.
- Include mapping macro IXGANSAA in your program. This macro shows the format of the answer area output returned for each system logger service in the ANSAREA parameter.
- If there are multiple connections to a log stream, each connected application must serialize delete requests so that a delete of log blocks does not occur, for example, in the middle of another application’s browse session.
- When coding the MODE=SYNCECB and ECB parameters, you must ensure that:
- The virtual storage area specified for the ECB resides on a full word boundary.
- You initialize the ECB field to zero.
- The ECB resides in either the common or home address space storage at the time the IXGDELET request is issued.
- The storage used for output parameters, such as ANSAREA and OBLOCKID, are accessible by both the IXGDELET invoker and the ECB waiter.
- When coding the MODE=SYNCEXIT parameter, you must ensure that the storage used for output parameters, such as ANSAREA and OBLOCKID, are accessible by both the IXGDELET invoker and the completion exit routine.

**Restrictions**

- All storage areas specified in this service must be in the same storage key as the caller's storage key and must exist in the caller's primary address space.
- There is more than one version of this macro available. The parameters you can use depend on the version you specify on the PLISTVER parameter. See the description of the PLISTVER parameter for more information.

**Input register information**

Before issuing the IXGDELET macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

**Output register information**

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if register 15 contains a non-zero return code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**Performance implications**

None.
### Syntax

The standard form of the IXGDELET macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td><code>name</code>: symbol. Begin <code>name</code> in column 1.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>One or more blanks must precede IXGDELET.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>IXGDELET</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>One or more blanks must follow IXGDELET.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>STREAMTOKEN=streamtoken</code></td>
<td><code>streamtoken</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>BLOCKS=ALL</code></td>
<td></td>
</tr>
<tr>
<td><code>BLOCKS=RANGE</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>BLOCKID=blockid</code></td>
<td><code>blockid</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ANSAREA=ansarea</code></td>
<td><code>ansarea</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ANSLEN=anslen</code></td>
<td><code>anslen</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>FORCE=NO</code></td>
<td><strong>Default</strong>: <code>FORCE=NO</code></td>
</tr>
<tr>
<td><code>FORCE=YES</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>FORCEINFO=NO</code></td>
<td><strong>Default</strong>: <code>FORCEINFO=NO</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>OBLOCKID=oblockid</code></td>
<td><code>oblockid</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>MODE=SYNC</code></td>
<td><strong>Default</strong>: <code>MODE=SYNC</code></td>
</tr>
<tr>
<td><code>MODE=ASYNCNORESPONSE</code></td>
<td></td>
</tr>
<tr>
<td><code>MODE=SYNCECB</code></td>
<td></td>
</tr>
<tr>
<td><code>MODE=SYNCEXIT</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>REQDATA=reqdata</code></td>
<td><code>reqdata</code>: RS-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ECB=ecb</code></td>
<td><code>ecb</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>DIAG=NO_DIAG</code></td>
<td>Default: DIAG=NO_DIAG</td>
</tr>
<tr>
<td><code>DIAG=NO</code></td>
<td></td>
</tr>
<tr>
<td><code>DIAG=YES</code></td>
<td></td>
</tr>
<tr>
<td><code>PLISTVER=IMPLIED_VERSION</code></td>
<td></td>
</tr>
<tr>
<td><code>PLISTVER=MAX</code></td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td><code>PLISTVER=0</code></td>
<td></td>
</tr>
<tr>
<td><code>PLISTVER=1</code></td>
<td></td>
</tr>
<tr>
<td><code>RETCODE=retcode</code></td>
<td><code>retcode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>RSNCODE=rsncode</code></td>
<td><code>rsncode</code>: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td><code>MF=S</code></td>
<td>Default: MF=S</td>
</tr>
<tr>
<td><code>MF=(L,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>MF=(L,list addr,attr)</code></td>
<td></td>
</tr>
<tr>
<td><code>MF=(L,list addr,0D)</code></td>
<td></td>
</tr>
<tr>
<td><code>MF=(E,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>MF=(E,list addr,COMPLETE)</code></td>
<td></td>
</tr>
<tr>
<td><code>MF=(E,list addr,NOCHECK)</code></td>
<td></td>
</tr>
<tr>
<td><code>MF=(M,list addr)</code></td>
<td></td>
</tr>
<tr>
<td><code>MF=(M,list addr,COMPLETE)</code></td>
<td></td>
</tr>
<tr>
<td><code>MF=(M,list addr,NOCHECK)</code></td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

- **`STREAMTOKEN=streamtoken`**
  Specifies the name or address (using a register) of a required 16-byte input field containing the token for the log stream that you want to search. The stream token is returned by the IXGCONN service at connection to the log stream.

- **`BLOCKS=ALL`**
- **`BLOCKS=RANGE`**
  Specifies whether all or just a subset of log blocks in a log stream be deleted.
  - BLOCKS=ALL: Specifies that all the log blocks in the specified log stream be deleted.
**IXGDELET macro**

- **BLOCKS=RANGE**: Specifies that the range of log blocks, older than the block specified on the BLOCKID parameter, be deleted. The BLOCKID parameter is required with BLOCKS=RANGE. See [z/OS MVS Programming: Assembler Services Guide](https://www.ibm.com/systems/z/os/zos/bkserv/) for more information on deleting a range of log blocks.

- **BLOCKID=blockid**
  Specifies the name or address (using a register) of an 8-byte input field which contains a log block identifier. BLOCKID is required with the BLOCKS=RANGE parameter. All blocks in the log stream older than the block specified on BLOCKID will be deleted. Note that the block specified in BLOCKID is not deleted.
  
  Block identifiers are returned in the RETBLOCKID field of the IXGWRITE service.

- **ANSAREA=ansarea**
  Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSA macro.

- **ANSLEN=anslen**
  Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.
  
  To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSA macro.

- **FORCE=NO**
  - **FORCE=YES**
    Specifies whether this delete request can be overridden by a resource manager exit.
    
    If you specify FORCE=NO, which is the default, the delete request can be overridden by the resource manager exit.
    
    If you specify FORCE=YES, the delete request cannot be overridden by a delete exit.

- **OBLOCKID=oblockid**
  Specifies the name or address (using a register) of an 8 character output field where the resource manager places the override block identifier.

- **MODE=SYNC**
  - **MODE=ASYNCCB**
  - **MODE=SYNCEXIT**
    Specifies that the request should be processed in one of the following ways:
    
    - **MODE=SYNC**: Specifies that the request process synchronously. Control is not returned to the caller until request processing is complete. If necessary, the calling program will be suspended until the request completes.
    
    - **MODE=ASYNCCB**: Specifies that the request process asynchronously. The caller is not notified when the request completes and the answer area (ANSAREA) fields will not contain valid information.
      
      To use this parameter, the system where the application is running must be IPLed. The application must run in supervisor state, key 0-7 to use this parameter.
    
    - **MODE=SYNCEXIT**: Specifies that the request process synchronously if possible. If the request processes asynchronously, control returns to the caller.
before the request completes and the event control block (ECB) specified on
the ECB parameter is posted when the request completes. The ECB
parameter is required with MODE=SYNCECB.

- MODE=SYNCEXIT: Specifies that the request process synchronously, if
  possible. If the request cannot be processed synchronously, your complete
  exit (specified on the COMPLETEEXIT parameter on the IXGCONN request)
  gets control when this request completes. Control returns to the caller with a
  return and reason code indicating that the request is not complete. The
  system passes the data specified on the REQDATA parameter, if specified, to
  the complete exit.

When a MODE=SYNCEXIT request processes asynchronously, system logger
maintains latent binds to the storage location specified by the answer area
(ANSAREA) fields, and, if specified, to RETBLOCKID and TIMESTAMP.

To use this parameter, the system where the application is running must be
IPLed. The application must run in supervisor state, key 0-7 to use this
parameters.

.REQDATA=reqdata
Specifications the name (or address in a register) of a 8-byte input field containing
user-defined data to pass to the complete exit. REQDATA is only valid with
the MODE=SYNCEXIT parameter.

ECB=ecb
Specifications the name or address (using a register) of a 4-byte input field that
contains an event control block (ECB) to be posted when the request
completes.

Before coding ECB, you must ensure that:
- You initialize the ECB.
- The ECB must reside in either common storage or the home address space
  where the IXGDELETE request was issued.
- The virtual storage area specified for the ECB must reside on a fullword
  boundary.

.DIAG=NO_DIA
.DIAG=NO
.DIAG=YES

Specifications whether or not the DIAG option on the IXGCONN for this logstream
will be in effect for this delete log data request. Refer to the DIAG keyword on
the IXGINVNT, IXGCONN and IXGBRWSE macro services.

If you specify DIAG=NO_DIA, which is the default, then the DIAG option
on the IXGCONN for this logstream will be in effect for this delete log data
request.

If you specify DIAG=NO, then Logger will not take additional diagnostic
action as defined on the logstream definition DIAG parameter.

If you specify DIAG=YES, then Logger will take additional diagnostic action as
defined on the logstream definition DIAG parameter providing the IXGCONN
connect DIAG specification allows it.

.PLISTVER=IMPLIED_VERSION
.PLISTVER=MAX
.PLISTVER=0
.PLISTVER=1

An optional input parameter that specifies the version of the macro. PLISTVER
determines which parameter list the system generates.
The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, supports all parameters except those specifically referenced in higher versions.

- **2**, supports both the following parameters and parameters from version 0:
  - FORCE
  - OBLOCKID
  - REQDATA

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0 or 1

\[ \text{RETCODE} = \text{retcode} \]

Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

\[ \text{RSNCODE} = \text{rsncode} \]

Specifies a name (or address in a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

\[ \text{MF} = \text{S} \]
\[ \text{MF} = \text{(L, list addr)} \]
\[ \text{MF} = \text{(L, list addr, attr)} \]
\[ \text{MF} = \text{(L, list addr, 0D)} \]
\[ \text{MF} = \text{(E, list addr)} \]
\[ \text{MF} = \text{(E, list addr, COMPLETE)} \]
\[ \text{MF} = \text{(E, list addr, NOCHECK)} \]
\[ \text{MF} = \text{(M, list addr)} \]
\[ \text{MF} = \text{(M, list addr, COMPLETE)} \]
\[ \text{MF} = \text{(M, list addr, NOCHECK)} \]

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.
IXGDELET macro

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:

- Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,list_addr,NOCHECK), to execute the macro.

\,list_addr
The name of a storage area to contain the parameters.

\,attr
An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code attr, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

\,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

\,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

ABEND codes
None.

Return and reason codes
When IXGDELET macro returns control to your program, GPR 15 contains a return code and GPR 0 contains a reason code.

Note: A program invoking the IXGDELET service may indicate through the MODE parameter that requests which can not be completed synchronously should have control returned to the caller prior to the completion of the request. When the request does complete, the invoker will be notified and the return and reason codes are in the answer area mapped by IXGANSAA.

The IXGCON macro provides equate symbols for the return and reason codes. The equate symbols associated with each hexadecimal return code are as follows:

00  IXGRETCODEOK - Service completes successfully.
04  IXGRETCODEWARNING - Service completes with a warning.
08  IXGRETCODEERROR - Service does not complete.
The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

### Table 119. Return and Reason Codes for the IXGDELET Macro

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>xxxx0000</td>
<td></td>
</tr>
</tbody>
</table>
|             |             | **Equate Symbol:** IxgRsnCodeOk  
|             |             | **Explanation:** Request processed successfully. |
| 04          | xxxx0401    | 
|             |             | **Equate Symbol:** IxgRsnCodeProcessedAsynch  
|             |             | **Explanation:** Program error. The program specified MODE=SYNCECB and the request must be processed asynchronously.  
|             |             | **Action:** Wait for the ECB specified on the ECB parameter to be posted, indicating that the request is complete. Check the ANSAA_ASYNCH_RETCODE and ANSAA_ASYNCH_RSN_CODE fields, mapped by IXGANSAA, to determine whether the request completed successfully. |
| 04          | xxxx040B    | 
|             |             | **Equate Symbol:** IxgRsnCodeRMNotConnected  
|             |             | **Explanation:** Program or environment error. The log stream is identified as being a source log stream managed by a resource manager (RMNAME is specified in the LOGR couple data set). However, at the time of the delete request, the resource manager was not connected to the log stream and FORCE=NO was specified on the request. Delete requests can only be honored on a resource manager managed system if the resource manager is connected to the log stream.  
|             |             | **Action:** Either:  
|             |             | • Start the resource manager so that it can connect to the log stream.  
|             |             | • Issue the IXGDELET request specifying FORCE=YES to delete the log block even though the resource manager is not connected to the source log stream. |
| 04          | xxxx040C    | 
|             |             | **Equate Symbol:** IxgRsnCodeROv errideOK  
|             |             | **Explanation:** The caller’s delete request was overridden by the associated resource manager. The override information was successfully processed. |
| 04          | xxxx040D    | 
|             |             | **Equate Symbol:** IxgRsnCodeRMNoBlock  
|             |             | **Explanation:** Program error. The log block identifier on the IXGDELET request does not exist in the log stream. Either the block id never existed or was deleted in a previous IXGDELET request. This warning is issued only if a resource manager overrides the caller-specified block id.  
|             |             | **Action:** Make sure that the block id specified on the IXGDELET request is correct. |
### IXGDELET macro

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04          | xxxx040E    | **Equate Symbol:** IxgRsnCodeRMBadGap  
**Explanation:** Environment error. The IXGDELET request failed because the requested log data was unreadable. This problem is caused by either an I/O error while attempting to read a DASD log data set or a log data set was deleted using an interface other than IXGDELET. This reason code is issued only when a resource manager exit overrides the block identifier specified on the IXGDELET request.  
**Action:** System logger returns the block identifier of the first readable log block (in the direction of youngest data) in the ANSAA_GAPS_NEXT_BLKID field of the answer area mapped by IXGANSSA. If appropriate, reissue the IXGDELET request using this block identifier. |
| 04          | xxxx040F    | **Equate Symbol:** IxgRsnCodeRMEOFGap  
**Explanation:** Environment error. While processing the IXGDELET request, system logger prematurely reached the end or beginning of the log stream. The portion of the log stream from the requested log data to either the beginning or end of the log stream was unreadable. This problem is caused by either an I/O error while attempting to read a DASD log data set or a log data set was deleted using an interface other than IXGDELET. This reason code is issued only when a resource manager exit overrides the block identifier specified on the IXGDELET request.  
**Action:** The action you take depends on whether your application can tolerate any loss of data. You can either:  
• Accept the loss of data and continue processing this log stream.  
• Stop using this log stream.  
• Correct the problem and re-issue the request. |
| 04          | xxxx0410    | **Equate Symbol:** IxgRsnCodeRMLossOfDataGap  
**Explanation:** Environment error. The log data you tried to delete is in a section of the log stream where data is permanently missing. This condition occurs when a system or coupling facility is in recovery from a failure and not all the log data could be recovered. This reason code is issued only when a resource manager exit overrides the block identifier specified on the IXGDELET request.  
**Action:** If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. If your application can tolerate data loss, you can continue using the log stream. |
| 04          | xxxx0411    | **Equate Symbol:** IxgRsnCodeRAMbended  
**Explanation:** Program error. The resource manager abended and percolated to the system logger recovery environment. The IXGDELET request was not processed.  
**Action:** Look for and correct the problem in your resource manager program or reissue the delete request, specifying FORCE=YES. |
Table 119. Return and Reason Codes for the IXGDELET Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04          | xxxx0412    | **Equate Symbol**: IxgRsnCodeRMDisabled  
**Explanation**: Environment error. The log stream is identified as being managed by a resource manager (RMNAME is specified in the LOGR couple data set). The resource manager is connected to the log stream, but is disabled due to an abend from which it did not recover successfully (by percolating to system logger recovery environment).  
**Action**: Either:  
* Cancel the resource manager exit and then restart the resource manager address space.  
* Reissue the request, specifying FORCE=YES. |
| 04          | xxxx0414    | **Equate Symbol**: IxgRsnCodeRMSoppedDelete  
**Explanation**: The resource manager does not allow this IXGDELET request to delete any log blocks.  
**Action**: Determine why the resource manager is prohibiting deletes. Specify FORCE=YES to stop the resource manager exit from stopping the delete request. |
| 08          | xxxx0801    | **Equate Symbol**: IxgRsnCodeBadParmlist  
**Explanation**: Program error. The parameter list could not be accessed.  
**Action**: Ensure that the storage area for the parameter list is accessible to the system logger for the duration of the request. The parameter list storage must be addressable in the caller's primary address space and in the same key as the caller. |
| 08          | xxxx0802    | **Equate Symbol**: IxgRsnCodeXESError  
**Explanation**: System error. A severe cross-system extended services (XES) error has occurred.  
**Action**: See ANSAA_DIAG1 for the XES return code and ANSAA_DIAG2 for the XES reason code. |
| 08          | xxxx0804    | **Equate Symbol**: IxgRsnCodeNoBlock  
**Explanation**: Program error. The block identifier or time stamp does not exist in the log stream. Either the value provided was never a valid location within the log stream or a prior IXGDELET request deleted the portion of the log stream it referenced.  
**Action**: Ensure that the value provided references an existing portion of the log stream and issue the request again. Use the LIST LOGSTREAM DETAIL(YES) request on the IXCMIPIU utility to display the range of valid block identifiers for the log stream. |
<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0806    | **Equate Symbol**: IxgRsnCodeBadStmToken  
 **Explanation**: Program error. One of the following occurred:  
• The stream token was not valid.  
• The specified request was issued from an address space other than the connector's address space.  
 **Action**: Do one of the following:  
• Make sure that the stream token specified is valid.  
• Ensure the request was issued from the connector's address space. |
| 08          | xxxx080A    | **Equate Symbol**: IxgRsnCodeRequestLocked  
 **Explanation**: Program error. The program issuing the request is holding a lock.  
 **Action**: Ensure that the program issuing the request is not holding a lock. |
| 08          | xxxx0814    | **Equate Symbol**: IxgRsnCodeNotAvailForIPL  
 **Explanation**: Environment error. The system logger address space is not available for the remainder of this IPL. The system issues messages about this error during system logger initialization.  
 **Action**: See the explanation for system messages issued during system logger initialization. |
| 08          | xxxx0815    | **Equate Symbol**: IxgRsnCodeNotEnabled  
 **Explanation**: Program error. The program issuing the request is not enabled for I/O and external interrupts, so the request fails.  
 **Action**: Make sure the program issuing the request is enabled for I/O and external interrupts. |
| 08          | xxxx0816    | **Equate Symbol**: IxgRsnCodeBadAnslen  
 **Explanation**: Program error. The answer area length (ANSLEN parameter) is not large enough. The system logger returned the required size in the Ansaa_Preferred_Size field of the answer area, mapped by IXGANSAA macro.  
 **Action**: Re-issue the request, specifying an answer area of the required size. |
| 08          | xxxx0817    | **Equate Symbol**: IxgRsnCodeBadAnsarea  
 **Explanation**: Program error. The storage area specified on the ANSAREA parameter cannot be accessed. This may occur after the system logger address space has terminated.  
 **Action**: Specify storage that is in the caller's primary address space and in the same key as the calling program at the time the system logger service was issued. This storage must be accessible until the request completes. |
### IXGDELET macro

#### Table 119. Return and Reason Codes for the IXGDELET Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx081C    | **Equate Symbol:** IxgRsnCodeNotAuthFunc  
**Explanation:** Program error. The program connected to the log stream with the AUTH=READ parameter and then tried to delete or write data. You cannot write or delete data when connected with read authority.  
**Action:** Issue the IXGCONN service with AUTH=WRITE authority and then re-issue this request. |
| 08          | xxxx081F    | **Equate Symbol:** IxgRsnCodeXcdsError  
**Explanation:** System error. System logger encountered an internal problem while processing the LOGR couple data set.  
**Action:** Contact the IBM Support Center. Provide the return and reason code and the contents of the answer area (ANSAREA field). |
| 08          | xxxx082D    | **Equate Symbol:** IxgRsnCodeExpiredStmToken  
**Explanation:** Environment error. The stream token is no longer valid because the connector has been disconnected.  
**Action:** Connect to the log stream again before issuing any functional requests. |
| 08          | xxxx0836    | **Equate Symbol:** IxgRsnCodeBadGap  
**Explanation:** Environment error. The request failed because the requested log data was unreadable. This condition could be caused by either an I/O error while attempting to read a log data set or a log data set deleted without using the IXGDELET interface.  
**Action:** The block identifier of the first accessible block toward the youngest data in the log stream is returned in the ANSAA_GAPS_NEXT_BLKID field in the answer area mapped by the IXGANSAA macro. If appropriate, re-issue the IXGDELET request using this block identifier. |
| 08          | xxxx083D    | **Equate Symbol:** IxgRsnCodeBadECBStor  
**Explanation:** Program error. The ECB storage area was not accessible to the system logger.  
**Action:** Ensure that the storage area is accessible to the system logger for the duration of the request. The storage must be addressable in the caller’s home address space and in the same key as the caller. |
### IXGDELET macro

**Table 119. Return and Reason Codes for the IXGDELET Macro (continued)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx084A    | **Equate Symbol:** IxgRsnCodeEOFGap  
**Explanation:** Environment error. The request prematurely reached the beginning or the end of the log stream. The portion of the log stream from the requested log data to either the beginning or the end of the log stream (depending on the direction of the read) was unreadable. This condition may be caused by either an I/O error while trying to read a log data set, or a log data set deleted without using the IXGDELET interface.  
**Action:** The action necessary is completely up to the application depending on how critical your data is. You can do one of the following:  
- Accept this condition and continue reading.  
- Stop processing the log all together.  
- Attempt to get the problem rectified, if possible, and then try to re-issue the request. |
| 08          | xxxx084B    | **Equate Symbol:** IxgRsnCodeLossOfDataGap  
**Explanation:** Environment error. The requested log data referenced a section of the log stream where log data is permanently missing. This condition occurs when a system or coupling facility is in recovery due to a failure, but not all of the log data in the log stream could be recovered.  
**Action:** If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss. |
| 08          | xxxx0861    | **Equate Symbol:** IxgRsnCodeRebuildInProgress  
**Explanation:** Environment error. No requests can be processed for this log stream because a coupling facility structure re-build is in progress for the structure associated with this log stream.  
**Action:** Listen for ENF signal 48 that will indicate one of the following:  
- The log stream is available because the re-build completed successfully. Re-issue the request.  
- The re-build failed and the log stream is not available. |
| 08          | xxxx0862    | **Equate Symbol:** IxgRsnCodeXESPurge  
**Explanation:** Environment error. An cross-system extended services (XES) request has been purged due to re-build processing.  
**Action:** Listen for ENF signal 48 that will indicate one of the following:  
- The log stream is available because the re-build completed successfully. Re-issue the request.  
- The re-build failed and the log stream is not available. |
<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0863    | Equate Symbol: IxgRsnCodeStructureFailed  
Explanation: Environment error. Either the coupling facility structure associated with the log stream has failed or the coupling facility itself has failed.  
Action: Listen for ENF signal 48 that will indicate one of the following:  
• The log stream is available because the re-build completed successfully. Re-issue the request.  
• The re-build failed and the log stream is not available. |
| 08          | xxxx0864    | Equate Symbol: IxgRsnCodeNoConnectivity  
Explanation: Environment error. No connectivity exists to the coupling facility associated with the log stream. The system logger will either attempt to re-build the log stream in another coupling facility or the log stream will be disconnected.  
Action: Listen for ENF signal 48 that will indicate one of the following:  
• The log stream is available because the re-build completed successfully. Re-issue the request.  
• The re-build failed and the log stream is not available.  
• The log stream has been disconnected from this system.  
If a re-build initiated because of a loss of connectivity previously failed, an ENF corresponding to this reason code might not be issued. Further action by the installation might be necessary to cause the change of the log stream status again. Check the log for messages IXG101I, IXG107I and related rebuild messages for information on resolving any outstanding issues. |
| 08          | xxxx0890    | Equate Symbol: IxgRsnCodeAddrSpaceNotAvail  
Explanation: System error. The system logger address space failed and is not available.  
Action: Do not issue system logger requests. |
| 08          | xxxx0891    | Equate Symbol: IxgRsnCodeAddrSpaceInitializing  
Explanation: System error. The system logger address space is not available because it is IPLing.  
Action: Listen for ENF signal 48, which will indicate when the system logger address space is available. Re-connect to the log stream, then re-issue this request. You can also listen for ENF signal 48, which will indicate if the system logger address space will not be available for the life of the IPL. In that case, do not issue system logger services. |
| 08          | xxxx08D0    | Equate Symbol: IxgRsnCodeProblemState  
Explanation: Environment error. The request was rejected because of one of the following:  
• The request was issued in SRB mode while the requestor was in problem program state.  
• The SYNCEXIT parameter was specified while the requestor’s PSW key was in problem program key.  
Action: Change the invoking environment to supervisor state. |
Table 119. Return and Reason Codes for the IXGDELET Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx08D1    | **Equate Symbol**: IxgRsnCodeProgramKey  
**Explanation**: Environment error. The request was rejected because of one of the following:  
• The request was issued in SRB mode while the requestor was in problem program key (key 8-F).  
• The SYNCEXIT parameter was specified while the requestor's PSW key was in problem program key.  
**Action**: Change the invoking environment to a system key (key 0-7). |
| 08          | xxxx08D2    | **Equate Symbol**: IxgRsnCodeNoCompleteExit  
**Explanation**: Program error. MODE=SYNCEXIT was specified, but the connection request did not identify a complete exit.  
**Action**: Either change this request to a different MODE option, or reconnect to the log stream with a complete exit specified on the COMPLETEEXIT parameter. |
| 08          | xxxx085F    | **Equate Symbol**: IxgRsnPercToRequestor  
**Explanation**: Environment error. Percolation to the service requestor's task occurred because of an abend during system logger processing. Retry was not allowed.  
**Action**: Issue the request again. If the problem persists, contact the IBM Support Center. |
| 0C          | xxxx0000    | **Equate Symbol**: IxgRetCodeCompError  
**Explanation**: User or System error. One of the following occurred:  
• You issued the FORCE IXGLOGR,ARM command to terminate the system logger address space.  
• System logger component error occurred.  
**Action**: If this reason code is not the result of forcing the system logger address space, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. Provide the diagnostic data in the answer area (IXGANSAA) and any dumps or LOGREC entries from system logger. |

Examples

Example 1: Delete all data from the log stream.

```assembly
IXGDELET STREAMTOKEN=TOKEN,                    X
  BLOCKS=ALL,                                  X
  MODE=SYNC,                                   X
  ANSAREA=ANSAREA,                             X
  ANSLEN=ANSLEN,                               X
  RSNCODE=RSNCODE,                             X
  MF=S,                                       X
  RETCODE=RETCODE                               X

ANSLEN DC A(L'ANSAREA)  length of logger's answer area
TOKEN DS CL16          stream token from connect
ANSAREA DS CL(ANSAA_LEN) answer area for log requests
RETCODE DS F            return code
RSNCODE DS F            reason code
DATAREA DSECT IXGANSAA LIST=YES       answer area
```
Example 2: Delete a range of data from the log stream asynchronously, if synchronous processing is not possible.

```assembly
IXGDELETE STREAMTOKEN=TOKEN, X
    BLOCKS=.Range, X
    BLOCKID=BLOCKID, X
    MODE=SYNCECB, X
    ECB=ANECB, X
    ANSAREA=ANSAREA, X
    ANSLEN=ANSLEN, X
    RSNCODE=RSNCODE, X
    MF=S, X
    RETCODE=RETCODE
```

* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
* | If rsncode = '00000401'X then wait on
* | the ecb ANECB.
* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

```assembly
ANSLEN DC A(L'ANSAREA) length of logger's answer area
BLOCKID DS CL8 block id from which to delete
TOKEN DS CL16 stream token from connect
ANSAREA DS CL(ANSAA_LEN) answer area for log requests
ANECB DS F ecb on which to wait
RETCODE DS F return code
RSNCODE DS F reason code
```

Example 3: Delete all data from the log stream using registers with the macro.

```assembly
LA R6,TOKEN load stream token into register 6
IXGDELETE STREAMTOKEN=(6), X
    BLOCKS=ALL, X
    MODE=SYNC, X
    ANSAREA=ANSAREA, X
    ANSLEN=ANSLEN, X
    RSNCODE=RSNCODE, X
    MF=S, X
    RETCODE=RETCODE
```

```assembly
ANSLEN DC A(L'ANSAREA) length of logger's answer area
TOKEN DS CL16 stream token from connect
ANSAREA DS CL(ANSAA_LEN) answer area for log requests
RETCODE DS F return code
RSNCODE DS F reason code
```

IXGDELET macro
IXGDELET macro
Chapter 134. IXGWRITE — Write log data to a log stream

Description

Use the IXGWRITE macro to allow a program to write a log block to a log stream. IXGWRITE returns a unique identifier for each log block written to the log stream.

System logger generates a time stamp for each log block as they are received from applications issuing IXGWRITE and writes the blocks to the log stream in that order. Applications that imbed their own time stamps in log blocks will find that the blocks may not be in application-generated time stamp order, especially if multiple applications are writing to a log stream simultaneously. In order to ensure chronological order of log blocks by application-generated time stamp, applications should provide their own serialization on the log stream.

For information on using the system logger services and the LOGR policy, see z/OS MVS Programming: Assembler Services Guide, which also includes information about related macros IXGCONN, IXGRBRWSE, IXGINVNT, IXGDELETE, and IXGQUERY.

Environment

The requirements for the caller are:

<table>
<thead>
<tr>
<th>Environmental factor</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum authorization:</td>
<td>Problem state with any PSW key. The caller must be</td>
</tr>
<tr>
<td></td>
<td>supervisor state with any system (0-7) PSW key to either</td>
</tr>
<tr>
<td></td>
<td>invoke this service in SRB mode or to use the</td>
</tr>
<tr>
<td></td>
<td>MODE=SYNCEXIT keyword.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>Any PASN, any HASN, any SASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit or 64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary or access register (AR)</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>No locks held</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All control parameters must be in the primary address space</td>
</tr>
<tr>
<td></td>
<td>with the following exceptions:</td>
</tr>
<tr>
<td></td>
<td>• The ECB should be addressable from the home address space.</td>
</tr>
</tbody>
</table>
|                            |   • Any parameter area that is explicitly ALET-qualified as allowed by the input parameter (for example, the area referenced by the BUFFER parameter when the BUFFALET parameter is specified) must be in an address or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All storage areas specified must be in the same storage key as the caller, with the following exception:  
|                            |     The parameter area is explicitly storage key qualified as allowed by the input parameters (example: the area referenced by the BUFFER parameter when the BUFFKEY parameter is also specified). |
IXGWRITE macro

Programming requirements

- Before issuing IXGWRITE, you must put the data you wish to write to the log stream into a buffer specified on the BUFFER parameter. IXGWRITE will then write this buffer to the log stream as a log block.
- The current primary address space from which you issue the IXGWRITE service must be the same as the primary address space at the time you issued the IXGCONN request.
- The parameter list for this service must be addressable in the caller's primary address space.
- The calling program must be connected to the log stream with write authority through the IXGCONN service.
- IXGWRITE cannot be issued if the connection is an import connection (IMPORTCONNECT=YES on the IXGCONN service). The IXGWRITE service must be issued under a write connection (IMPORTCONNECT=NO, which is the default).
- Include the IXGCON mapping macro in your program. This macro provides a list of equate symbols for the system logger services.
- Include mapping macro IXGANSAA in your program. This macro shows the format of the answer area output returned for each system logger service in the ANSAREA parameter.
- When coding the MODE=SYNCECB and ECB parameters, you must ensure that:
  - The virtual storage area specified for the ECB resides on a full word boundary.
  - You initialize the ECB field to zero.
  - The ECB resides in either the common or home address space storage at the time the IXGWRITE request is issued.
  - The storage used for output parameters, such as ANSAREA, RETBLOCKID, and TIMESTAMP, are accessible by both the IXGWRITE invoker and the ECB waiter.
- When coding the MODE=SYNCEXIT parameter, you must ensure that the storage used for output parameters, such as ANSAREA, RETBLOCKID, and TIMESTAMP, are accessible by both the IXGWRITE invoker and the completion exit routine.

Restrictions

- All storage areas specified on this macro must be in the same storage key as the caller's storage key, with the exception of the BUFFKEY parameter.
  Storage areas that are not ALET-qualified must exist in the caller's primary address space. The ECB should be addressable from the home address space.
- There is more than one version of this macro available. The parameters you can use depend on the version you specify on the PLISTVER parameter. See the description of the PLISTVER parameter for more information.
- You can call any of the system logger services in either AMODE 31 or 64, but the parameter list and all other data addresses, with the exception of BUFFER64 must reside in 31-bit storage.

Input register information

Before issuing the IXGWRITE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.
Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code, if register 15 contains a non-zero return code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IXGWRITE macro is written as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name: symbol. Begin name in column 1.</td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must precede IXGWRITE.</td>
</tr>
<tr>
<td>IXGWRITE</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow IXGWRITE.</td>
</tr>
<tr>
<td>,STREAMTOKEN=streamtoken</td>
<td>streamtoken: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BUFFER=buffer</td>
<td>buffer: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>BUFFER64=buffer64</td>
<td>buffer64: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BLOCKLEN=blocklen</td>
<td>blocklen: RS-type address or register (2) - (12).</td>
</tr>
</tbody>
</table>
### IXGWRITE macro

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,RETBLOCKID=retblockid</td>
<td>retblockid: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ANSAREA=ansarea</td>
<td>ansarea: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ANSLEN=anslen</td>
<td>anslen: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BUFFKEY=buffkey</td>
<td>buffkey: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,BUFFKEY=*</td>
<td>Default: BUFFKEY=*</td>
</tr>
<tr>
<td>,BUFFALET=buffalet</td>
<td>buffalet: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,TIMESTAMP=timestamp</td>
<td>timestamp: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>MODE=SYNC</td>
<td>Default: MODE=SYNC</td>
</tr>
<tr>
<td>MODE=ASYNCNORESPONSE</td>
<td></td>
</tr>
<tr>
<td>MODE=SYNCECB</td>
<td></td>
</tr>
<tr>
<td>MODE=SYNCEXIT</td>
<td></td>
</tr>
<tr>
<td>,REQDATA=reqdata</td>
<td>reqdata: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ECB=ecb</td>
<td>ecb: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,PLISTVER=IMPLIED_VERSION</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>,PLISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=plistver</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=0</td>
<td></td>
</tr>
<tr>
<td>,PLISTVER=1</td>
<td></td>
</tr>
<tr>
<td>,RETCODE=retcode</td>
<td>retcode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,RSNCODE=rsncode</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,MF=S</td>
<td>Default: MF=S</td>
</tr>
<tr>
<td>,MF=(L,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,attr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(L,list addr,0D)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr)</td>
<td></td>
</tr>
<tr>
<td>,MF=(E,list addr,COMPLETE)</td>
<td></td>
</tr>
</tbody>
</table>
### Syntax Description

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MF=(E,list addr,NOCHECK)$</td>
<td></td>
</tr>
<tr>
<td>$MF=(M,list addr)$</td>
<td></td>
</tr>
<tr>
<td>$MF=(M,list addr,COMPLETE)$</td>
<td></td>
</tr>
<tr>
<td>$MF=(M,list addr,NOCHECK)$</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

The parameters are explained as follows:

- **STREAMTOKEN=streamtoken**
  Specifies the name (or address in a register) of a required 16-byte input field containing the token for the log stream that you want to write to. The stream token is returned by the IXGCONN service at connection to the log stream.

- **BUFFER=buffer**
  - **BUFFER64=buffer64**
    Specifies the field name (or address in a register) of the data to be written to the log.
    - BUFFER=buffer specifies that the location of the buffer is in 31-bit storage.
    - BUFFER64=buffer64 specifies that the location of the buffer is in 64-bit storage.

  The BUFFER and BUFFER64 parameters are mutually exclusive.

- **BLOCKLEN=blocklen**
  Specifies the name (or address in a register) of a 4-byte input field that contains the length in bytes of the log block you are writing to the log stream.
  The value of BLOCKLEN must be between 1 and the value for MAXBUFSIZE.

- **RETBLOCKID=retblockid**
  Specifies the name (or address in a register) of a 8-byte output field where IXGWRITE returns the unique block identifier for the log block written to the log stream.

- **ANSAREA=ansarea**
  Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

- **ANSLEN=anslen**
  Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

  To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

- **BUFFALET=buffalet**
  Specifies the name (or address in a register) of a 4-byte input field specifying the access list entry table (ALET) to be used to access the buffer specified on the BUFFER or BUFFER64 keyword. If the buffer is ALET-qualified, the ALET must index a valid entry on the task's dispatchable unit access list (DUAL) or specify a SCOPE=COMMON data space. An ALET that indexes the system logger PASN-AL list will not work.
The default is 0, which means that the buffer is in the calling program's primary address space.

**BUFFKEY=**buffkey
Specifies the name (or address in a register) of a 4-byte input field specifying the storage key for the buffer specified on the BUFFER or BUFFER64 parameter.

If the caller is running in problem state, the caller's PSW key and the key specified in the BUFFKEY parameter must match.

If the caller is running in supervisor state, specify any syntactically valid (0 through 15) key on the BUFFKEY parameter.

If you omit the BUFFKEY parameter, the default used is the PSW key of the caller.

**TIMESTAMP=**timestamp
Specifies the name (or address in a register) of a 16-byte output field where the Greenwich mean time and local time stamps associated with the requested log block are returned when the write request is successful. Both time stamps will be in time of day (TOD) clock format.

**MODE=**SYNC
**MODE=**ASYNCRponse
**MODE=**SYNCECB
**MODE=**SYNCEXIT

Specifies that the request should be processed in one of the following ways:

- **MODE=**SYNC: Specifies that the request process synchronously. Control is not returned to the caller until request processing is complete. If necessary, the calling program will be suspended until the request completes.

- **MODE=**ASYNCRponse: Specifies that the request process asynchronously. The caller is not notified when the request completes and the answer area (ANSAREA) fields will not contain valid information.

  To use this parameter, the system where the application is running must be IPLed. The application must run in supervisor state, key 0-7 to use this parameter.

- **MODE=**SYNCECB: Specifies that the request process synchronously if possible. If the request processes asynchronously, control returns to the caller before the request completes and the event control block (ECB) specified on the ECB keyword is posted when the request completes. The ECB keyword is required with **MODE=**SYNCECB.

- **MODE=**SYNCEXIT: Specifies that the request process synchronously, if possible. If the request cannot be processed synchronously, your complete exit (specified on the COMPLETEEXIT parameter on the IXGCON request) gets control when this request completes. Control returns to the caller with a return and reason code indicating that the request is not complete. The system passes the data specified on the REQDATA parameter, if specified, to the complete exit.

  When a **MODE=**SYNCEXIT request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to RETBLOCKID and TIMESTAMP.

  To use this parameter, the system where the application is running must be IPLed. The application must run in supervisor state, key 0-7 to use this parameter.
IXGWRITE macro

,REQDATA=\texttt{reqdata}

Specifies the name (or address in a register) of a 8-byte input field containing user-defined data to pass to the complete exit. REQDATA is only valid with the MODE=SYNCEXIT parameter.

,ECB=\texttt{ecb}

Specifies the name (or address in a register) of a 4-byte input field that contains the event control block (ECB) to be posted when the request completes.

Before coding ECB, you must ensure that:

- You initialize the ECB to zero.
- The ECB must reside in either common storage or the home address space where the IXGWRITE service was issued.
- The virtual storage area specified for the ECB must reside on a fullword boundary.

,PLISTVER=\texttt{IMPLIED\_VERSION}
,PLISTVER=\texttt{MAX}
,PLISTVER=\texttt{0}
,PLISTVER=\texttt{1}

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- \texttt{IMPLIED\_VERSION}, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED\_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.
- \texttt{MAX}, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- 0, which supports all parameters except those specifically referenced in higher versions.
- 1, which supports both the following parameters and parameters from version 0:
  - \texttt{REQDATA}

To code: Specify in this input parameter one of the following:

- \texttt{IMPLIED\_VERSION}
- \texttt{MAX}
- A decimal value of 0 or 1

,RETCODE=\texttt{retcode}

Specifies a name (or address in a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=\texttt{rsncode}

Specifies a name (or address in a register) of a 4-byte output field where the
system will place the reason code. The reason code is also in general purpose
register (GPR) 0, if you received a non-zero return code.

MF=S
MF=(L,list addr)
MF=(L,list addr,attr)
MF=(L,list addr,0D)
MF=(E,list addr)
MF=(E,list addr,COMPLETE)
MF=(E,list addr,NOCHECK)
MF=(M,list addr)
MF=(M,list addr,COMPLETE)
MF=(M,list addr,NOCHECK)

Use MF=S to specify the standard form of the macro, which builds an inline
parameter list and generates the macro invocation to transfer control to the
service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with
the execute form of the macro for applications that require reentrant code. The
list form defines an area of storage that the execute form uses to store the
parameters. Only the PLISTVER parameter can be specified on the list form of
the macro. IBM recommends that you always specify PLISTVER=MAX on the
list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form
together with the list form of the macro for applications that require reentrant
code. The execute form of the macro stores the parameters into the storage area
defined by the list form, and generates the macro invocation to transfer control
to the service.

Use MF=M together with the list and execute forms of the macro for service
routines that need to provide different options according to user-provided
input. Use the list form to define a storage area; use the modify form to set the
appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following
order:
  • Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters,
    including all required ones.
  • Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to
    change.
  • Use MF=(E,list_addr,NOCHECK), to execute the macro.

list addr
The name of a storage area to contain the parameters.

attr
An optional 1- to 60-character input string, which can contain any value
that is valid on an assembler DS pseudo-op. You can use this parameter to
force boundary alignment of the parameter list. If you do not code attr, the
system provides a value of 0D, which forces the parameter list to a
doubleword boundary.

COMPLETE
Specifies that the system is to check for required parameters and supply
defaults for omitted optional parameters.

NOCHECK
Specifies that the system is not to check for required parameters and is not
to supply defaults for omitted optional parameters.
ABEND codes

None.

Return and reason codes

When IXGWRITE macro returns control to your program, GPR 15 contains a return code and GPR 0 contains a reason code.

Note: A program invoking the IXGWRITE service may indicate through the MODE parameter that requests which can not be completed synchronously should have control returned to the caller prior to completion of the request. When the request does complete, the invoker will be notified and the return and reason codes are in the answer area mapped by IXGANSAA.

The IXGCON macro provides equate symbols for the return and reason codes. The equate symbols associated with each hexadecimal return code are as follows:

- **00** IXGRSNCODEOK - Successful completion.
- **04** IXGRSNCODEWARNING - The request was processed successfully, however a warning condition was encountered.
- **08** IXGRETCODEERROR - An error has been encountered. The associated reason code provides more information.
- **0C** IXGRETCODECOMPEMOR - A system logger component error has been encountered.

The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

### Table 120. Return and Reason Codes for the IXGWRITE Macro

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>xxxx0000</td>
<td>Equate Symbol: IxgRsnCodeOk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Explanation: Request processed successfully.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0401</td>
<td>Equate Symbol: IxgRsnCodeProcessedAsynch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Explanation: Program error. The program specified MODE=SYNCECB and the request must be processed asynchronously.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Wait for the ECB specified on the ECB parameter to be posted, indicating that the request is complete. Check the ANSAA ASYNCH RETCODE and ANSAA ASYNCH RSNCODE fields, mapped by IXGANSAA, to determine whether the request completed successfully.</td>
</tr>
</tbody>
</table>
### Table 120. Return and Reason Codes for the IXGWRITE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>xxxx0405</td>
<td>Equate Symbol: IxgRsnCodeWarningLossOfData</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation:</strong> Environment error. Returned for READCURSOR, START OLDEST and RESET OLDEST requests. This condition occurs when a system and coupling facility fail and not all of the log data in the log stream could be recovered.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For READCURSOR: A log block has been returned, but there may be log blocks permanently missing between this log block and the one previously returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For START OLDEST and RESET OLDEST: The oldest log blocks in the log stream may be permanently missing, the browse cursor is set at the oldest available log block.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0407</td>
<td>Equate Symbol: IxgRsnCodeConnPossibleLossOfData</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation:</strong> Environment error. The request was successful, but there may be log blocks permanently missing between this log block and the one previously returned. This condition occurs when a system or coupling facility fails and not all of the data in the log stream could be recovered.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0408</td>
<td>Equate Symbol: IxgRsnCodeDsDirectoryFullWarning</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation:</strong> Environment error. The request was successful, but the log streams DASD data set directory is full. System logger cannot offload any further data from the coupling facility structure to DASD. The system logger will continue to process IXGWRITE requests until this log streams portion of the coupling facility structure becomes full.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Either delete enough data from the log stream to free up space in the log streams data set directory so that offloading can occur or disconnect from the log stream.</td>
</tr>
<tr>
<td>04</td>
<td>xxxx0409</td>
<td>Equate Symbol: IxgRsnCodeWowWarning</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Explanation:</strong> Environment error. The request was successful, but an error condition was detected during a previous offload of data. System logger might not be able to offload further data. System logger will continue to process IXGWRITE requests only until the interim storage for the log stream is filled. (Interim storage is the coupling facility for a coupling facility log stream and local storage buffers for a DASD-only log stream.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Do not issue any further requests for this log stream and disconnect. Connect to another log stream. Check the system log for message IXG301I to determine the cause of the error. If you cannot fix the error, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</td>
</tr>
</tbody>
</table>
### Table 120. Return and Reason Codes for the IXGWRITE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04          | 0000040A    | **Equate Symbol:** IxgRsnCodeDuplexFailureWarning  
**Explanation:** Environment error. The request was successful, but the system logger was unable to duplex log data to staging data sets, even though the log stream definition requested unconditional duplexing to staging data sets by specifying the log stream attributes: STG_DUPLEX=YES, DUPLEXMODE=UNCOND, or STG_DUPLEX=YES,DUPLEXMODE=DRXRC. When DUPLEXMODE=UNCOND is specified, but Logger was unable to obtain a staging data set to duplex the log data. Therefore, the Logger duplexing is being done in local buffers (data space).  
When DUPLEXMODE=DRXRC is specified for a logstream and being used for (non-local) disaster recovery duplexing, if the internal buffers used for asynchronous buffering of the log blocks become full. Meaning the internal buffers became full before at least one of the full buffers could be written to the staging data set.  
**Action:** For DUPLEXMODE=UNCOND, if duplexing to staging data sets is required, disconnect from this log stream and connect to a log stream that can be duplexed to staging data sets.  
For DUPLEXMODE=DRXRC, if duplexing to a DRXRC-type staging data sets is required, then cause the log data to be offload to the log stream secondary storage (offload data sets) and then continue writing to the log stream. |
| 08          | xxxx0801    | **Equate Symbol:** IxgRsnCodeBadParmlist  
**Explanation:** Program error. The parameter list could not be accessed.  
**Action:** Ensure that the storage area for the parameter list is accessible to the system logger for the duration of the request. The parameter list storage must be addressable in the caller's primary address space and in the same key as the caller. |
| 08          | xxxx0802    | **Equate Symbol:** IxgRsnCodeXESError  
**Explanation:** System error. A severe cross-system extended services (XES) error has occurred.  
**Action:** See ANSAA_DIAG1 for the XES return code and ANSAA_DIAG2 for the XES reason code. |
| 08          | xxxx0803    | **Equate Symbol:** IxgRsnCodeBadBuffer  
**Explanation:** Program error. The virtual storage area specified on the BUFFER or BUFFER64 parameter is not addressable.  
**Action:** Ensure that the storage area specified on the BUFFER or BUFFER64 parameter is addressable to system logger for the duration of the request. If the BUFFKEY parameter is specified, make sure it contains a valid key associated with the storage area. If BUFFKEY is not used, ensure that the storage is in the same key as the program at the time the logger service was requested. The storage must be addressable in the caller's primary address space.
### Table 120. Return and Reason Codes for the IXGWRITE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0806    | **Equate Symbol:** IxgRsnCodeBadStmToken  
**Explanation:** Program error. One of the following occurred:  
• The stream token was not valid.  
• The specified request was issued from an address space other than the connector's address space.  
**Action:** Do one of the following:  
• Make sure that the stream token specified is valid.  
• Ensure the request was issued from the connector's address space. |
| 08          | xxxx0809    | **Equate Symbol:** IxgRsnCodeBadWriteSize  
**Explanation:** Program error. The size of the log block specified in the BLOCKLEN parameter is not valid. The value for BLOCKLEN must be greater than zero and less than or equal to the maximum buffer size (MAXBUFSIZE) defined in the LOGR policy for the structure associated with this log stream.  
**Action:** Ensure that the value specified on the BLOCKLEN parameter is greater than 0 and less than or equal to the MAXBUFSIZE which is returned on the log stream connect request. |
| 08          | xxxx080A    | **Equate Symbol:** IxgRsnCodeRequestLocked  
**Explanation:** Program error. The program issuing the request is holding a lock.  
**Action:** Ensure that the program issuing the request is not holding a lock. |
| 08          | xxxx0814    | **Equate Symbol:** IxgRsnCodeNotAvailForIPL  
**Explanation:** Environment error. The system logger address space is not available for the remainder of this IPL. The system issues messages about this error during system logger initialization.  
**Action:** See the explanation for system messages issued during system logger initialization. |
| 08          | xxxx0815    | **Equate Symbol:** IxgRsnCodeNotEnabled  
**Explanation:** Program error. The program issuing the request is not enabled for I/O and external interrupts, so the request fails.  
**Action:** Make sure the program issuing the request is enabled for I/O and external interrupts. |
| 08          | xxxx0816    | **Equate Symbol:** IxgRsnCodeBadAnslen  
**Explanation:** Program error. The answer area length (ANSLEN parameter) is not large enough. The system logger returned the required size in the Ansaa_Preferred_Size field of the answer area, mapped by IXGANSAA macro.  
**Action:** Re-issue the request, specifying an answer area of the required size.
### IXGWRITE macro

#### Table 120. Return and Reason Codes for the IXGWRITE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0817    | **Equate Symbol:** IxgRsnCodeBadAnsarea  
**Explanation:** Program error. The storage area specified on the ANSAREA parameter cannot be accessed. This may occur after the system logger address space has terminated.  
**Action:** Specify storage that is in the caller's primary address space and in the same key as the calling program at the time the system logger service was issued. This storage must be accessible until the request completes. |
| 08          | xxxx0818    | **Equate Symbol:** IxgRsnCodeBadBlockidStor  
**Explanation:** Program error. The storage area specified by BLOCKID cannot be accessed.  
**Action:** Ensure that the storage area is accessible to system logger for the duration of the request. The storage must be addressable in the caller's primary address space and in the same key as the caller. |
| 08          | xxxx081C    | **Equate Symbol:** IxgRsnCodeNotAuthFunc  
**Explanation:** Program error. The program connected to the log stream with the AUTH=READ parameter and then tried to delete or write data. You cannot write or delete data when connected with read authority.  
**Action:** Issue the IXGCONN service with AUTH=WRITE authority and then re-issue this request. |
| 08          | xxxx082D    | **Equate Symbol:** IxgRsnCodeExpiredStmToken  
**Explanation:** Environment error. The stream token is no longer valid because the connector has been disconnected.  
**Action:** Connect to the log stream again before issuing any functional requests. |
| 08          | xxxx0837    | **Equate Symbol:** IxgRsnCodeBadTimestamp  
**Explanation:** Program error. The storage area specified by TIMESTAMP cannot be accessed.  
**Action:** Ensure that the storage area is accessible to the system logger service for the duration of the request. The storage must be addressable in the caller's primary address space and in the same key as the caller. |
| 08          | xxxx083D    | **Equate Symbol:** IxgRsnCodeBadECBStor  
**Explanation:** Program error. The ECB storage area was not accessible to the system logger.  
**Action:** Ensure that the storage area is accessible to the system logger for the duration of the request. The storage must be addressable in the caller's home address space and in the same key as the caller. |
| 08          | xxxx083F    | **Equate Symbol:** IxgRsnCodeTestartError  
**Explanation:** System error. An unexpected error was encountered while attempting to validate the buffer ALET.  
**Action:** See ANSAA_DIAG1 in the answer area mapped by the IXGANSAA macro for the return code from the TESTART system service. |
### Table 120. Return and Reason Codes for the IXGWRITE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0841    | **Equate Symbol**: IXGErrCodeBadBufferALET  
**Explanation**: Program error. The buffer ALET specified is not zero and does not represent a valid entry on the caller’s dispatchable unit access list (DUAL). See the ANSAA_DIAG1 field of the answer area, mapped by the IXGANSAA macro, for the return code from the TESTART system service.  
**Action**: Ensure that the correct ALET was specified. If not, provide the correct ALET. Otherwise, add the correct ALET to dispatchable unit access list (DUAL). |
| 08          | xxxx0849    | **Equate Symbol**: IXGErrCodeBadBuffkey  
**Explanation**: Program error. The buffer key specified on the BUFFKEY parameter specifies an invalid key. Either the key is greater than 15 or the program is running in problem state and the specified key is not the same key as the PSW key at the time the system logger service was issued.  
**Action**: For problem state programs, either do not specify the BUFFKEY parameter or else specify the same key as the PSW key at the time the system logger service was issued. For supervisor state programs, specify a valid storage key (0 <= key <= 15). |
| 08          | xxxx085C    | **Equate Symbol**: IXGErrCodeDsDirectoryFull  
**Explanation**: Environment error. The interim storage (for example: the coupling facility structure space allocated or the staging data set space) for the log stream is full. System logger’s attempts to offload the interim storage log data to DASD has failed because the log stream’s data set directory is full. If this reason code is issued by the IXGWRITE request, no further write requests can be processed until additional directory space is available for the log stream.  
System logger will periodically re-drive its offload attempts for this condition, which is applicable to both coupling facility structure and DASD-only type log streams. If system logger is able to offload log data, then an ENF event will be issued informing the connectors that the log stream should be available for writing more log data. However, the time that passes before you can write to the log stream is unpredictable.  
The system issues related messages IXG257I, IXG261E, IXG262A and IXG301I.  
**Action**: The system programmer must make more log stream data set directory space available.  
For information about how an authorized application program might respond to this reason code, see Setting Up the System Logger Configuration in the z/OS MVS Programming: Authorized Assembler Services Guide.  
For information about how an unauthorized application program might respond to this reason code, see IXGWRITE: Writing to a log stream in the z/OS MVS Programming: Assembler Services Guide. |
### Table 120. Return and Reason Codes for the IXGWRITE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx085D    | **Equate Symbol:** IxgRsnCodeWowError  
**Explanation:** Environment error. The interim storage (for example: the coupling facility structure space allocated or the staging data set space) for the log stream is full. System logger’s attempts to offload the interim storage log data to DASD have failed because of severe errors. No further write requests can be processed until the offload error condition is cleared.  
System logger will periodically re-drive its offload attempts for this condition, which is applicable to both coupling facility structure and DASD-only type log streams. If system logger is able to offload log data, then an ENF event will be issued informing the connectors that the log stream should be available for writing more log data. However, the time that passes before you can write to the log stream is unpredictable.  
The system issues related message IXG301I.  
**Action:** The system programmer must correct the severe error condition inhibiting the log stream offload. If you are unable to correct the error, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.  
You can retry your write request periodically or wait for the ENF signal that the log stream is available, or disconnect from this log stream and connect to another log stream.  
For information on how an authorized application program might respond to this reason code, see Setting up the system logger configuration in the z/OS MVS Programming Authorized Assembler Services Guide.  
For information on how an authorized application program might respond to this reason code, see IXGWRITE: Writing to a log stream in the z/OS MVS Programming: Assembler Services Guide. |
| 08          | xxxx0860    | **Equate Symbol:** IxgRsnCodeCFLogStreamStorFull  
**Explanation:** Environment error. The coupling facility structure space allocated for this log stream is full. No further requests can be processed until the log data in the coupling facility structure is offloaded to DASD log data sets.  
**Action:** Listen to the ENF signal 48 which will indicate that the log stream is available after the data has been offloaded to DASD. For IXGCONN requests, Listen to the ENF signal 48 which will indicate that the structure is available. Then, re-issue the request. |
**Table 120. Return and Reason Codes for the IXGWRITE Macro (continued)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0861    | **Equate Symbol**: IxgRsnCodeRebuildInProgress  
**Explanation**: Environment error. No requests can be processed for this log stream because a coupling facility structure re-build is in progress for the structure associated with this log stream.  
**Action**: Listen for ENF signal 48 that will indicate one of the following:  
- The log stream is available because the re-build completed successfully. Re-issue the request.  
- The re-build failed and the log stream is not available. |
| 08          | xxxx0862    | **Equate Symbol**: IxgRsnCodeXESPurge  
**Explanation**: Environment error. An cross-system extended services (XES) request has been purged due to re-build processing.  
**Action**: Listen for ENF signal 48 that will indicate one of the following:  
- The log stream is available because the re-build completed successfully. Re-issue the request.  
- The re-build failed and the log stream is not available. |
| 08          | xxxx0863    | **Equate Symbol**: IxgRsnCodeStructureFailed  
**Explanation**: Environment error. Either the coupling facility structure associated with the log stream has failed or the coupling facility itself has failed.  
**Action**: Listen for ENF signal 48 that will indicate one of the following:  
- The log stream is available because the re-build completed successfully. Re-issue the request.  
- The re-build failed and the log stream is not available. |
| 08          | xxxx0864    | **Equate Symbol**: IxgRsnCodeNoConnectivity  
**Explanation**: Environment error. No connectivity exists to the coupling facility associated with the log stream. The system logger will either attempt to re-build the log stream in another coupling facility or the log stream will be disconnected.  
**Action**: Listen for ENF signal 48 that will indicate one of the following:  
- The log stream is available because the re-build completed successfully. Re-issue the request.  
- The re-build failed and the log stream is not available.  
- The log stream has been disconnected from this system.  
If a re-build initiated because of a loss of connectivity previously failed, an ENF corresponding to this reason code might not be issued. Further action by the installation might be necessary to cause the change of the log stream status again. Check the log for messages IXG101I, IXG107I and related rebuild messages for information on resolving any outstanding issues. |
## IXGWRITE macro

### Table 120. Return and Reason Codes for the IXGWRITE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0865    | **Equate Symbol:** IxgRsnCodeStagingDSFull  
**Explanation:** Environment error. The staging data set allocated for this log stream on this system is full. No further requests can be processed until enough log data in the coupling facility structure is offloaded to DASD log data sets to relieve the staging data set's full condition.  
**Action:** Listen to the ENF signal 48 which will indicate that the log stream is available after room becomes available in the staging data set. Then, re-issue the request. |
| 08          | xxxx0867    | **Equate Symbol:** IxgRsnCodeLocalBufferFull  
**Explanation:** Environment error. One of the two following problems was detected:  
- The available local buffer space (data space storage) for the system logger address space is full. ANSAA_Diag1 and ANSAA_Diag2 in the Answer Area will contain 0 for this error return.  
- The IXGWRITE is rejected because a caller attempted to write log data when the outstanding asynchronous write activity for this connection was considered too high. The limit for unauthorized IXGWRITE invokers is 2,000 and the limit of 10,000 is used for authorized callers. An unauthorized caller is a caller whose PSW key is >= 8 and that is not in supervisor state. ANSAA_DIAG1 in the answer area contains a value of 1 for this error return for unauthorized callers and a value of 2 for authorized callers. ANSAA_DIAG2 contains the total number of outstanding write requests for this connection.  
No further writing requests can be processed until the log data in the local buffer space is offloaded to DASD log data sets or this connector's prior IXGWRITE requests complete.  
**Note:** This reason code applies to both CF and DASD only log stream requests.  
**Action:**  
- For authorized writers: Listen for the ENF signal 48 that will indicate that the log stream is available. With the first condition, logger issues the ENF signal after the data has been offloaded to DASD. With the second condition, logger issues the ENF signal 48 that the log stream is available when the number of in-flight authorized asynchronous writes is reduced below 85% of the limit. There will be no ENF signal issued when the unauthorized limit is relieved.  
- For unauthorized callers: Wait for a short interval and then reissue the request.  
- If the attempts continue to fail or the ENF signal is not issued for an unacceptable period, consider notifying operations or disconnecting from the log stream. |
### Table 120. Return and Reason Codes for the IXGWRITE Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08          | xxxx0868    | **Equate Symbol:** IxgRsnCodeStagingDSFormat  
**Explanation:** Environment error. The staging data set allocated for this log stream on this system has not finished being formatted for use by System Logger. No further IXGWRITE requests can be processed until the formatting completes.  
**Action:** Listen to the ENF signal 48 which will indicate that the logstream is available after formatting process is finished. Then, re-issue the request. |
| 08          | xxxx0890    | **Equate Symbol:** IxgRsnCodeAddrSpaceNotAvail  
**Explanation:** System error. The system logger address space failed and is not available.  
**Action:** Do not issue system logger requests. |
| 08          | xxxx0891    | **Equate Symbol:** IxgRsnCodeAddrSpaceInitializing  
**Explanation:** System error. The system logger address space is not available because it is IPLing.  
**Action:** Listen for ENF signal 48, which will indicate when the system logger address space is available. Re-connect to the log stream, then re-issue this request. You can also listen for ENF signal 48, which will indicate if the system logger address space will not be available for the life of the IPL. In that case, do not issue system logger services. |
| 08          | xxxx08D1    | **Equate Symbol:** IxgRsnCodePrgramKey  
**Explanation:** Environment error. The request was rejected because of one of the following:  
- The request was issued in SRB mode while the requestor was in problem program key (key 8-F).  
- The SYNCEXIT parameter was specified while the requestor's PSW key was in problem program key.  
**Action:** Change the invoking environment to a system key (key 0-7). |
| 08          | xxxx08D2    | **Equate Symbol:** IxgRsnCodeNoCompleteExit  
**Explanation:** Program error. MODE=SYNCEXIT was specified, but the connection request did not identify a complete exit.  
**Action:** Either change this request to a different MODE option, or reconnect to the log stream with a complete exit specified on the COMPLETEEXIT parameter. |
| 08          | xxxx08D7    | **Equate Symbol:** IxgRsnCodeRequestNotAllowed  
**Explanation:** Program error. The caller issued an IXGWRITE request while an import connection was active on this system (IXGCONN IMPORTCONNECT=YES).  
**Action:** Re-issue the request, based on the type of connection active. |
### Example 1

Write data to the log stream synchronously.

```
IXGWRITE STREAMTOKEN=TOKEN,
    BUFFER=BUFF, X
    BLOCKLEN=BLKLEN,
    BUFALET=BUFALET,
    RETBLOCKID=RETBLK,
    BUFSIZE=BUFSIZE,
    TIMESTAMP=RET_TIME,
    MODE=SYNC,
    ANSAREA=ANSAREA,
    ANSLEN=ANSLEN,
    RSNCODE=RSNCODE,
    MF=S,
    RETCODE=RETCODE
```

<table>
<thead>
<tr>
<th>BUFF</th>
<th>DC CL256 'BUFFER TEXT'</th>
<th>buffer to write to log stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLKLEN</td>
<td>DC F'256'</td>
<td>length of block to be written</td>
</tr>
<tr>
<td>ANSLEN</td>
<td>DC A(L'ANSAREA)</td>
<td>length of logger's answer area</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>DC F'8'</td>
<td>buffer key</td>
</tr>
<tr>
<td>TOKEN</td>
<td>DS CL16</td>
<td>stream token from connect</td>
</tr>
<tr>
<td>RET_TIME</td>
<td>DS CL16</td>
<td>returned timestamp of block</td>
</tr>
<tr>
<td>ANSAREA</td>
<td>DS CL(ANSAA_LEN)</td>
<td>answer area for log requests</td>
</tr>
<tr>
<td>RETCODE</td>
<td>DS F</td>
<td>return code</td>
</tr>
<tr>
<td>RSNCODE</td>
<td>DS F</td>
<td>reason code</td>
</tr>
<tr>
<td>BUFALET</td>
<td>DC F'1'</td>
<td>buffer alet secondary</td>
</tr>
<tr>
<td>RETBLK</td>
<td>DS CLB</td>
<td>returned block id</td>
</tr>
</tbody>
</table>

### Example 2

Write data to the log stream asynchronously, if synchronous processing is not possible.

```
IXGWRITE STREAMTOKEN=TOKEN,
    BUFFER=BUFF, X
    BLOCKLEN=BLKLEN,
    BUFALET=BUFALET,
    RETBLOCKID=RETBLK,
    MODE=SYNCECB,
    ECB=ANECB,
    ANSAREA=ANSAREA,
    ANSLEN=ANSLEN,
    RSNCODE=RSNCODE,
    MF=S,
    RETCODE=RETCODE
```

| IXGANSAA | DSECT IXGANSAA LIST=YES | answer area |

---

**Table 120. Return and Reason Codes for the IXGWRITE Macro (continued)**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C</td>
<td>xxxxx0000</td>
<td><strong>Equate Symbol</strong>: IxgRetCodeCompError</td>
</tr>
</tbody>
</table>

**Explanation**: User or System error. One of the following occurred:

- You issued the FORCE IXGLOGR,ARM command to terminate the system logger address space.
- System logger component error occurred.

**Action**: If this reason code is not the result of forcing the system logger address space, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. Provide the diagnostic data in the answer area (IXGANSAA) and any dumps or LOGREC entries from system logger.
IXGWRITE macro

* if return code = '00000401'X then wait
* on the ecb ANECB for the request to complete
BUFF DC CL256'BUFFER TEXT' buffer to write to log stream
BLKLEN DC F'256' length of block to be written
ANSLEN DC A(L'ANSAREA) length of logger's answer area
TOKEN DS CL16 stream token from connect
ANSAREA DS CL(ANSAA_LEN) answer area for log requests
RETCODE DS F return code
RSNCODE DS F reason code
BUFALET DC F'1' buffer alet secondary
ANECB DS F ecb to wait on
RETBLK DS CL8 returned block id
DATAREA DSECT
IXGANSAA LIST=YES answer area

Example 3
Write data to the log stream using registers.
LA R6,TOKEN load stream token in register 6
   IXGWRITE STREAMTOKEN=(6), X
   BUFFER=BUFF, X
   BLOCKLEN=BLKLEN, X
   RETBLOCKID=RETBLK, X
   MODE=SYNC, X
   ANSAREA=ANSAREA, X
   ANSLEN=ANSLEN, X
   RSNCODE=RSNCODE, X
   MF=S, X
   RETCODE=RETCODE
BUFF DC CL256'BUFFER TEXT' buffer to write to log stream
BLKLEN DC F'256' length of block to be written
ANSLEN DC A(L'ANSAREA) length of logger's answer area
TOKEN DS CL16 stream token from connect
ANSAREA DS CL(ANSAA_LEN) answer area for log requests
RETCODE DS F return code
RSNCODE DS F reason code
RETBLK DS CL8 returned block id
DATAREA DSECT
   IXGANSAA LIST=YES answer area
R6 EQU 6 set up register 6
Appendix. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- z/OS TSO/E Primer
- z/OS TSO/E User's Guide
- z/OS ISPF User's Guide Vol I

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out
punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 \* FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* \* FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the
default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1 (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

* indicates an optional syntax element that is repeatable
The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:
1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included
The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.
Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel  
IBM Corporation  
2455 South Road  
Poughkeepsie, NY 12601-5400  
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted.
for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

**Minimum supported hardware**

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS](http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM representative.

**Programming interface information**

This information is intended to help the customer to code macros that are available to authorized assembler language programs. This information documents intended programming interfaces that allow the customer to write programs to obtain services of z/OS.

**Trademarks**

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [Copyright and Trademark information](http://www.ibm.com/legal/copytrade.shtml).

UNIX is a registered trademark of The Open Group in the United States and other countries.
## Index

### Numerics

31-bit addressing mode  
- macros requiring expansion  
  ESPAE macro 99

64-bit  
- guidance information 342, 465

64-bit latch  
- obtaining 1217  
- purging 1229  
- purging a group 1233  
- purpose 1217, 1223, 1229, 1233  
- releasing 1223

64-bit latch manager services  
- ISGLPB64 callable service  
  syntax 1233

64-bit latch set  
- creating 1201  
- purpose 1201

A  
- abend  
  interrupting scheduled 99  
- accessibility 1397  
- contact IBM 1397  
- features 1397

addressing mode and the services 2  
- ALET qualification of parameters 4

AR () mode  
- description 3

ASC (address space control) mode  
- defining 3  
- assistive technologies 1397

C  
- callable service  
  coding 16  
  captured UCB  
  obtaining actual address 921  
  coding the callable services 16  
  coding the macros 13  
  component trace format table  
  generating 1267  
- contact  
  z/OS 1397

continuation line 15

D  
- data sharing with IARSVENV macro 329  
- device measurement block index  
  obtaining 901

E  
- EDT (eligible device table)  
  obtaining information 27

EDTINFO macro 27  
- ENF (event notification facility)  
  example 43

ENFREQ macro 35  
- ENQ macro 71  
- entry table  
  connecting 117  
  creating 123  
  destroying 137  
  disconnecting 143

- entry table descriptor  
  creating 127

ESPIE macro 89  
- ESTAE and ESTAEX macros 99  
- ETCON macro 117  
- ETCRE macro 123  
- ETDEF macro 127  
- ETDES macro 137  
- ETDIS macro 143

- event  
  notification of occurrence 35  
  waiting for completion 147  
- EVENTS macro 147  
- EXTRACT macro 153

F  
- FESTAE macro 159  
- FRACHECK macro (for RACF Release 1.8.1 or earlier) 165  
- FREEMAIN macro 167

G  
- GETDSAB macro 179  
- GETMAIN macro 187  
- global serialization queue  
  extracting information 205  
- GQSCAN macro 205  
- GTRACE macro 219  
  DATA function 223  
  QUERY function 221  
  TEST function 219

H  
- hyperspace  
  reading to 259  
  writing from 259

- HISMT macro 229  
- HISSERV macro 259  
- HSPSERV macro 259

I  
- I/O configuration token  
  obtaining 901  
- IARCP64 macro 279  
- IARR2V macro 297  
- IARST64 macro 303  
- IARSUBSP macro 319  
- IARV64 macro 341

- REQUEST=CHANGEACCESS 441  
- REQUEST=DETACH 420  
- REQUEST=GETSHARED 429  
- REQUEST=SHAREMEMOBJ 435

- IARSVENV macro  
  data sharing 329

- IAZXCTKN macro 471  
- IAZXJSAB macro 473

- IEAAPBE callable service 685  
- IEAAPBE2 callable service 689

- IEADPE callable service 697

- IEADPE2 callable service 701

- IEAIPSE callable service 705  
- IEAIPSE2 callable service 711  
- IEAIRLS callable service 717  
- IEAIRLS2 callable service 723

- IEARPI callable service 729

- IEARPI2 callable service 735

- IEATPE callable service 743

- IEAXFR callable service 747

- IEAXFR2 callable service 755

- IEAARR macro 479

- IEAFP macro 485

- IEALSRDY macro 491

- IEAMETR macro 495

- IEAMRFR3 macro 499

- IEAMSCHD macro 507

- IEAN4CR callable service 541

- IEAND4L callable service 547

- IEAN4RT callable service 551

- IEANTCR callable service 523

- IEANTDL callable service 529

- IEANTRT callable service 533

- IEARBUP RB Update Service 555

- IEATEDS macro 585

- IEATXDC macro 607

- IEAVAPE callable service 611

- IEAVAPE2 callable service 615

- IEAVDPE callable service 623

- IEAVDPE2 callable service 627

- IEAVPSE callable service 633

- IEAVPSE2 callable service 639

- IEAVRDL callable service 645

- IEAVRSL2 callable service 651

- IEAVRPI callable service 657

- IEAVRPI2 callable service 663

- IEAVTPE callable service 669

- IEAVXFR callable service 673

- IEAVXFR2 callable service 679

- IEECMDS macro 761

- IEETXCS macro 771

- IEEVERS macro 787

- comparison to MGCRE macro 787

- IEFPSPSCN macro 801

- IEFQREQ macro 809

- IEFJSYS macro 813

- Environment 813

- Input register information 814

© Copyright IBM Corp. 1988, 2015

1405
IEFSJSYM macro (continued)
Output register information 814
Performance implications 814
Programming requirements 814
REQUEST= parameter 815
Parameters 816
Restrictions 814
Return and reason codes 818
IEFSSI macro 823
IEFSSVT macro 859
IEFSSVTI macro 881
IFAQUERY service 895
IOCINFO macro 901
building control unit entry 985
ENQ 993
FBA (fixed block architecture) 999
obtain switch information 1079
obtaining information 957, 969, 979, 1025, 1063
IOSADMF macro 909
IOSCAPF macro 921
IOSCAPU macro 925
IOSCDR macro 943
IOSCHPD macro 957
IOSCMBA macro 965
IOSCMXOA macro 969
IOSCMXRA macro 975
IOSCLNF macro 979
IOSCMUOF macro 985
 IOSDCXR macro 989
IOSENQ macro 993
IOSFBA macro 999
IOSHXBLK macro 1017
IOSINOF macro 1025
IOSLOOK macro 1029
IOSODS macro 1031
Execute Form 1035
List Form 1034
IOSPFTH macro 1039
IOSPSPOF macro 1049
Execute Form 1061
List Form 1060
IOSUPFA macro 1063
IOSUPFSR macro 1067
IOSVRYSW macro 1071
IOSWITCH macro 1079
IOSWIFHPF macro 1085
IQUPINFO macro 1089
IRDPSD macro 1095
IRDPSDU macro 1101
ISERCA macro 1130
ISGLCR64 callable service 1201
syntax 1201
ISGLCRT callable service 1165
syntax 1165
ISGLOB64 callable service 1217
syntax 1217
ISGLOBT callable service 1181
syntax 1181
ISGLPR64 callable service 1233
syntax 1197
ISGLPBA callable service 1197
syntax 1197
ISGLPPG callable service 1193
syntax 1193
ISGLRE64 callable service 1223
syntax 1223
ISGLREL callable service 1187
syntax 1187
ITTMTM macro 1267
ITTWRIE macro 1273
ITZXFILT macro 1281
IXGBRWS macro 1285
IXGCNN macro 1333
IXGDELET macro 1359
IXGWRITE macro 1377
job
obtaining information about current 473
keyboard
navigation 1397
PF keys 1397
shortcut keys 1397
latch
obtaining 1181
purging 1193
purging a group 1197
purpose 1181, 1187, 1193, 1197
releasing 1187
latch manager services
ISGLCR64 callable service
syntax 1201
ISGLCRT callable service
syntax 1165
ISGLOB64 callable service
syntax 1217
ISGLOBT callable service
syntax 1181
ISGLPBA callable service
syntax 1197
ISGLPR64 callable service
syntax 1223
ISGLPPG callable service
syntax 1229
ISGLRE64 callable service
syntax 1193
ISGLREL callable service
syntax 1187
latch set
creating 1165
purpose 1165
linkage stack query macro 491
macro (continued)
level
selecting 1
sample 14
selecting level 1
user parameter, passing 4
X-macros
using 11
navigation
keyboard 1397
Notices 1401
sending comments to IBM xxxi
serially reusable resource requesting control 71
service
ALET qualification 4
summary 17
services
addressing mode 2
ASC mode defining 3
using 1
sharing storage with IARVSERV macro 329
shortcut keys 1397
SMF
configuration query service 895
subchannel number
obtaining for a UCB 1025
Summary of changes xxxiii
SWA manager
invoking in move mode 809
TCB (task control block)
extracting information 153
unit control block
capturing and releasing 925
locating 1029
obtaining address of common extension segment 975
obtaining address of device class extension 989
obtaining address of prefix extension 1067
vary switch service 1071
user interface
ISPF 1397
TSO/E 1397
user parameter
passing 4
V
virtual storage
   allocating 167, 187
Virtual storage
   sharing with IARVSERV macro 329

X
X-macros
   using 11
Product Number: 5650-ZOS

Printed in USA