

z/OS



MVS Programming: Callable Services for High-Level Languages

Version 2 Release 2

Note

Before using this information and the product it supports, read the information in "Notices" on page 607.

This edition applies to Version 2 Release 2 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 1994, 2016.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures vii

Tables ix

About this information xi

Who should use this information xi

How to use this information xi

z/OS information xi

How to send your comments to IBM xiii

If you have a technical problem xiii

Summary of changes xv

Summary of changes for z/OS Version 2 Release 2,
as updated June 2016 xv

Summary of changes for z/OS Version 2 Release 2,
as updated March, 2016 xv

Summary of changes for z/OS Version 2 Release 2,
as updated December, 2015 xv

Summary of changes for z/OS Version 2 Release 2 xvi

Summary of changes for z/OS Version 2 Release 1,
as updated September, 2014 xvii

Summary of changes for z/OS Version 2 Release 1,
as updated March, 2014 xvii

z/OS Version 2 Release 1 summary of changes xvii

Part 1. Window services. 1

Chapter 1. Introduction to window services. 3

Permanent data objects 3

Temporary data objects 3

Structure of a data object 3

What does window services provide? 4

 The ways that window services can map an object 5

 Access to permanent data objects 8

 Access to temporary data objects 9

Chapter 2. Using window services 11

Obtaining access to a data object 12

 Identifying the object 12

 Specifying the object's size 13

 Specifying the type of access 13

 Obtaining a scroll area 13

Defining a view of a data object 14

 Identifying the data object 14

 Identifying a window 14

 Defining the disposition of a window's contents 15

 Defining the expected reference pattern 15

 Identifying the blocks you want to view 16

 Extending the size of a data object 17

Defining multiple views of an object 17

 Non-overlapping views 17

 Overlapping views 17

Saving interim changes to a permanent data object 18

Updating a temporary data object 18

Refreshing changed data 19

Updating a permanent object on DASD 20

 When there is a scroll area 20

 When there is no scroll area 20

Changing a view in a window 20

Terminating access to a data object 22

Handling return codes and abnormal terminations 22

Chapter 3. Window services 23

CSREVV — View an object and sequentially access

it 23

 Abend codes 25

 Return codes and reason codes 26

CSRIDAC — Request or terminate access to a data

object 27

 Abend codes 30

 Return codes and reason codes 30

CSRREFR — Refresh an object 31

 Abend codes 32

 Return codes and reason codes 33

CSRSAVE — Save changes made to a permanent

object 34

 Abend codes 35

 Return codes and reason codes 35

CSRSCOT — Save object changes in a scroll area 36

 Abend codes 37

 Return codes and reason codes 37

CSRVIEW — View an object 39

 Abend codes 41

 Return codes and reason codes 41

Chapter 4. Window services coding examples. 45

ADA example 45

C/370 example 50

COBOL example 53

FORTRAN example 57

Pascal example 61

PL/I example 65

Part 2. Reference pattern services 71

Chapter 5. Introduction to reference pattern services. 73

How does the system manage data? 73

An example of how the system manages data in an

array 74

 What pages does the system bring in when a gap

exists? 76

Chapter 6. Using reference pattern services 79

Defining the reference pattern for a data area 79
 Defining the range of the area 79
 Identifying the direction of the reference. 80
 Defining the reference pattern 80
 Choosing the number of bytes on a page fault. 82
Examples of using CSRIRP to define a reference pattern 83
Removing the definition of the reference pattern 84
Handling return codes. 85

Chapter 7. Reference pattern services 87

CSRIRP — Define a reference pattern. 87
 Return codes and reason codes 89
CSRRRP — Remove a reference pattern 89
 Return codes and reason codes 90

Chapter 8. Reference pattern services coding examples 91

C/370 example 91
COBOL example. 94
FORTRAN example 98
Pascal example 101
PL/I example 103

Part 3. Global resource serialization latch manager services 107

Chapter 9. Using the latch manager services. 109

Syntax and linkage conventions for latch manager callable services 109
ISGLCRT — Create a latch set. 110
 ABEND codes 112
 Return codes 112
 Examples of calls to latch manager services 112
ISGLOBT — Obtain a latch. 114
 ABEND codes 116
 Return codes 116
 Example 117
ISGLREL — Release a latch. 117
 ABEND codes 119
 Return codes 119
 Example 120
ISGLPRG — Purge a requestor from a latch set 120
 ABEND codes 121
 Return codes 121
 Example 122
ISGLPBA — Purge a group of requestors from a group of latch sets. 122
 ABEND codes 124
 Return codes 124

Part 4. Resource recovery services (RRS) 125

Chapter 10. Using protected resources 127

Resource recovery programs 127
Two-phase commit protocol 128
Resource recovery process 128
Requesting resource protection and recovery 131
Using distributed resource recovery 131
Application_Backout_UR (SRRBACK) 132
 Description 132
Application_Commit_UR (SRRCMIT) 136
 Description 136
Additional callable services. 140

Part 5. CEA TSO/E address space services 141

Chapter 11. Introduction to CEA TSO/E address space services 143

CEA TSO/E address space manager components 143
System prerequisites for the CEA TSO/E address space services 144
Working with TSO/E address spaces started by CEA 145
 Communicating with programs running in the TSO/E address spaces 146
 Reconnecting to CEA TSO/E address spaces 148

Chapter 12. Using CEA TSO/E address space services. 151

Invoking the CEATsoRequest API 151
 Parameters 152
 Requirements for callers. 157
Understanding the request types 157
 CeaTsoStart - Starting a new TSO/E session 157
 CeaTsoAttn - Sending an attention interrupt to a TSO/E session 158
 CeaTsoEnd - Ending a TSO/E session 159
 CeaTsoPing - Sending a ping on behalf of an application 160
 CeaTsoQuery - Querying the TSO/E address spaces 160
 CeaTsoQueryApp - Querying TSO/E sessions by application 161
Return, reason, and diagnostic codes 162
 Return codes 163
 Reason codes 163
 Diagnostic codes 167
CEAYTSOR header file 170
CEAXRDEF header file 174
Programming example 179
 Sample compile job 196

Part 6. zEnterprise Data Compression (zEDC) 197

Chapter 13. Overview and planning of zEnterprise Data Compression (zEDC) 199

Requirements for zEnterprise Data Compression 200

Planning for zEnterprise Data Compression 200

Chapter 14. Application interfaces for zEnterprise Data Compression 203

Invoking unauthorized interfaces for zEnterprise Data Compression. 203
 zlib for zEnterprise Data Compression 203
Invoking System z authorized interfaces for zEnterprise Data Compression. 207
 System z authorized compression services. 208

Chapter 15. Troubleshooting for zEDC 223

Part 7. Other callable services. 225

Chapter 16. IEAAFFN — Assign processor affinity for encryption or decryption. 227

Restrictions and limitations. 228
Requirements 228
Return codes 228

Chapter 17. CSRL16J — Transfer control to another routine 231

Defining the entry characteristics of the target routine 231
Freeing dynamic storage associated with the caller 232
Programming requirements. 232
Restrictions 234
Performance implications 234
Syntax diagram. 235
 C/370 syntax 235
 PL/I syntax 235
Parameters 235
Return codes 235
Example 236
 C/370 example program 236
 Assembler program for use with the C/370 example 238

Chapter 18. CSRSI — System information service 239

Description 239
 Environment 239
 Programming requirements. 239
 Restrictions 240
 Input register information 240
 Output register information 240
 Syntax. 240
 Parameters 241
 Return codes 242
CSRSIC C/370 header file 243

Part 8. Base Control Program internal interface (BCPii) services . 255

Chapter 19. Base Control Program internal interface (BCPii) 257

BCPii setup and installation 257
 Setting up connectivity to the support element 258
 Setting up authority to use BCPii. 262
 BCPii configuration considerations 264
 Setting up event notification for BCPii z/OS UNIX applications. 266
 Setting up access to BCPii REXX execs 268
 BCPii startup and shutdown 269
| SMF recording in BCPii 270
| BCPii callable services 270
Syntax, linkage and programming considerations 271
 Calling formats. 271
 BCPii connection scope 272
 Linkage considerations 272
 REXX programming considerations 272
 Assembler programming considerations . . . 281
 Programming Examples 281
HWICMD — Issue a BCPii hardware management command 282
 Description 282
HWICONN — Establish a BCPii connection . . . 300
 Description 301
HWIDISC — Release a BCPii connection 311
 Description 311
HWEVENT — Register or unregister for BCPii events 317
 Monitoring events occurring on a particular CPC or image 317
 Monitoring operating system message events (Hwi_Event_OpSysMsg). 317
 Monitoring communication availability between BCPii and the CPC 318
 Monitoring the status of the BCPii address space 318
 Description 319
HWILIST — Retrieve HMC and BCPii configuration-related information. 329
 Description 329
HWIQUERY — BCPii retrieval of SE/HMC-managed attributes 341
 Description 341
HWISET — BCPii set SE/HMC-managed attributes 370
 Description 370
HWIBeginEventDelivery — Begin delivery of BCPii event notifications. 399
 Description 400
HWIEndEventDelivery — End delivery of BCPii event notifications. 403
 Description 403
HWIManageEvents — Manage the list of BCPii events 406
 Description 406
HWIGetEvent — Retrieve outstanding BCPii event notifications 410
 Description 410

Part 9. z/OS client web enablement toolkit 417

Chapter 20. The z/OS JSON parser	419
Elements of the z/OS JSON parser	420
Availability of the z/OS JSON parser	421
Syntax, linkage, and programming considerations	421
z/OS JSON parser callable services	426
HWTCONST — Initialize predefined variables (REXX)	428
HWTJCREN — Create JSON entry	429
HWTJESCT — Encode or decode escape sequences (REXX)	438
HWTJGAEN — Get array entry	439
HWTJGBOV — Get boolean value	443
HWTJGJST — Get JSON type	447
HWTJGNUE — Get number of entries	451
HWTJGNUV — Get number value (non-REXX)	455
HWTJGOEN — Get object entry	460
HWTJGVAL — Get value	466
HWTJINIT — Initialize a parser instance	470
HWTJPARS — Parse a JSON string	474
HWTJSERI — Serialize (build) JSON text	479
HWTJSRCH — Search	485
HWTJTERM — Terminate a parser instance	492
Chapter 21. The z/OS HTTP/HTTPS protocol enabler	497
Elements of the z/OS HTTP/HTTPS enabler	498
Availability of the z/OS HTTP/HTTPS enabler . .	499
Syntax, linkage, and programming considerations	499
z/OS HTTP/HTTPS callable services	508
HWTCONST — Initialize predefined variables (REXX)	509
HWTHCONN — Connect to an HTTP server	510
HWTHDISC — Disconnect from an HTTP server	515
HWTHINIT — Initialize an HTTP connection or request	520
HWTHRQST — Send a request to an HTTP server	524
HWTHRSET — Reset an HTTP connection or request	530
HWTHSET — Set HTTP connection or request options	535
HWTHSLST — Linked list append service	541
HWTHTERM — Terminate an HTTP connection or request	547
HTTP/HTTPS enabler options and values	552
Response callback (exit) routines (non-REXX).	562

Part 10. SMF services.	565
Chapter 22. SMF real-time interface	567
IFAMCON — Connect to an SMF in-memory resource	567
IFAMDSC — Disconnect from an SMF in-memory resource	570
IFAMGET — Obtain data from an SMF in-memory resource	573
IFAMQRY — Query SMF in-memory resources . .	577

Part 11. Appendixes 581

Appendix A. BCPii communication error reason codes 583

Appendix B. BCPii summary tables **585**

HWIQUERY and HWISET	585
HWICMD	596
HWIEVENT	598

Appendix C. General use C/C++ header files 601

Appendix D. Accessibility 603

Accessibility features	603
Consult assistive technologies	603
Keyboard navigation of the user interface	603
Dotted decimal syntax diagrams	603

Notices 607

Policy for unsupported hardware.	608
Minimum supported hardware	609
Additional notices	609
Programming interface information	610
Trademarks	610

Glossary 611

Index 613

Figures

1. Structure of a Data Object	4	13. Backout — Application Request	130
2. Mapping a Permanent Object That Has No Scroll Area	5	14. Backout — Resource Manager Votes NO	131
3. Mapping a Permanent Object That Has a Scroll Area	6	15. Transaction — Distributed Resource Recovery	132
4. Mapping a Temporary Object	6	16. Sample REXX EXEC	145
5. Mapping an Object to Multiple Windows	7	17. Example illustrating that the REXX SYSTEMID is the same as the z/OSMF ISPF application identifier	146
6. Mapping Multiple Objects	8	18. Sample TSO/E messages written to the queue	148
7. Illustration of a Reference Pattern with a Gap	76	19. Contents included in the ceasapit.x file	151
8. Two Typical Reference Patterns	80	20. CSRLJPLI declarations for return codes for PL/I	234
9. Illustration of Forward Direction of Reference	81	21. BCPii setup and installation steps.	258
10. Illustration of Backward Direction of Reference	82	22. Example of JSON text.	420
11. ATM Transaction	129		
12. Two-Phase Commit Actions.	130		

Tables

1. CSREVIEW Return and Reason Codes	26	43. Parameters for the FPZ4ABC service	216
2. CSRIDAC Return and Reason Codes	30	44. Header elements in the FPZ4ABC-generated list	217
3. CSRREFR Return and Reason Codes	33	45. Entries elements in the FPZ4ABC-generated list	217
4. CSRSAVE Return and Reason Codes	35	46. Return and Reason Codes for the FPZ4ABC service	218
5. CSRSCOT Return and Reason Codes	38	47. Environment for the FPZ4URZ service	220
6. CSRVIEW Return and Reason Codes	42	48. Parameters for the FPZ4URZ service	220
7. ISGLCRT Return Codes	112	49. Return and Reason Codes for the FPZ4URZ service	220
8. ISGLOBT Return Codes	117	50. IEAAFFN Return Codes	228
9. ISGLREL Return Codes	119	51. CSRL16J Return Codes	235
10. ISGLPRG Return Codes	121	52. Minimum BCPii microcode levels by SE hardware level	259
11. ISGLPBA Return Codes	124	53. Minimum BCPii microcode levels by HMC level	259
12. CEA TSO/E address space manager components	143	54. Minimum BCPii microcode levels by LPAR level	260
13. System prerequisites	144	55. BCPii APIs supported in the REXX environment	273
14. Message type identifiers	146	56. HWIREXX keywords	274
15. Message types	147	57. Return codes from the HWIREXX service	274
16. Data types	147	58. Return codes from a REXX BCPii host command	279
17. Input and output for each structure used for the CeaTsoStart request type	158	59. REXX return codes from the BCPii hwhost function	280
18. Input and output for each structure used for the CeaTsoAttn request type	158	60. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) .	288
19. Input and output for each structure used for the CeaTsoEnd request type.	159	61. Reasons for abend X'042', RC X'0001yyyy'	295
20. Input and output for each structure used for the CeaTsoPing request type	160	62. Reasons for abend X'042', RC X'0002yyyy'	305
21. Input and output for each structure used for the CeaTsoQuery request type	161	63. Reasons for abend X'042', RC X'0003yyyy'	313
22. Input and output for each structure used for the CeaTsoQueryApp request type	162	64. Reasons for abend X'042', RC X'0004yyyy'	324
23. Return codes	163	65. Reasons for abend X'042', RC X'0005yyyy'	335
24. Reason codes	164	66. Reasons for abend X'042', RC X'0006yyyy'	365
25. Diagnostic code	167	67. Reasons for abend X'042', RC X'0007yyyy'	394
26. Comparison table between unauthorized and System z authorized interfaces for zEDC	201	68. Reasons for abend X'042', RC X'0004yyyy'	408
27. Standard zlib functions and whether they are supported using zEDC	204	69. JSON parser programming interface	421
28. Compression and decompression with zlib	207	70. Calling formats for the z/OS JSON parser callable services.	422
29. Compression and decompression with System z authorized interfaces for zEDC	208	71. Host return codes for REXX.	424
30. Environment for the FPZ4RZV service	209	72. JSON parser programming sample files	426
31. Parameters for the FPZ4RZV service	209	73. Return codes for the HWTCONST service	429
32. Return and reason codes for the FPZ4RZV service	210	74. Return codes for the HWTJCREN service	433
33. Environment for the FPZ4PRB service	211	75. Return codes for the HWTJESCT service	439
34. Parameters for the FPZ4PRB service	211	76. Return codes for the HWTJGAEN service	442
35. Return and Reason Codes for the FPZ4PRB service	211	77. Return codes for the HWTJGBOV service	445
36. Environment for the FPZ4RMR service	212	78. Return codes for the HWTJGJST service	449
37. Parameters for the FPZ4RMR service	212	79. Return codes for the HWTJGNUE service	453
38. Return and Reason Codes for the FPZ4RMR service	213	80. Return codes for the HWTJGNUV service	458
39. Environment for the FPZ4DMR service	214	81. Return codes for the HWTJGOEN service	463
40. Parameters for the FPZ4DMR service	214	82. Return codes for the HWTJGVAL service	468
41. Return and Reason Codes for the FPZ4DMR service	215	83. Return codes for the HWTJINIT service	472
42. Environment for the FPZ4ABC service	215	84. Return codes for the HWTJPARS service	477
		85. Return codes for the HWTJSERI service	482
		86. Return codes for the HWTJSRCH service	488

87. Return codes for the HWTJTERM service	494	99. Return codes for the HWTHSET service	538
88. HTTP enabler	500	100. Return codes for the HWTHSLST service	544
89. Calling formats for the z/OS HTTP enabler callable services.	500	101. Return codes for the HWTHTERM service	549
90. Toolkit handling of HTTP redirection status response codes	505	102. Return and reason codes for the IFAMCON service.	569
91. Host return codes for REXX.	506	103. Return and reason codes for the IFAMDSC service.	572
92. z/OS HTTP enabler programming sample files.	508	104. Return and reason codes for the IFAMGET service.	575
93. Return codes for the HWTCONST service	510	105. Return and reason codes for the IFAMQRY service.	579
94. Return codes for the HWTHCONN service	512	106. HWIQUERY and HWISET attributes	585
95. Return codes for the HWTHDISC service	517	107. HWICMD types	596
96. Return codes for the HWTHINIT service	523	108. HWIEVENT types	598
97. Return codes for the HWTHRQST service	526		
98. Return codes for the HWTHRSET service	532		

About this information

Callable services are for use by any program coded in C, COBOL, FORTRAN, Pascal, or PL/I — this information refers to programs written in these languages as high-level language (HLL) programs. Callable services enable HLL programs to use specific MVS™ services by issuing program CALLs.

Who should use this information

This information is for programmers who code in C, COBOL, FORTRAN, Pascal, or PL/I and want to use the callable services that MVS provides.

How to use this information

This information 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 V2R2 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 V2R2 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 documentation. Please provide us with any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

Use one of the following methods to send your comments:

Important: If your comment regards a technical problem, see instead “If you have a technical problem.”

- Send an email to mhvrcfs@us.ibm.com.
- Send an email from the "Contact us" web page for z/OS (<http://www.ibm.com/systems/z/os/zos/webqs.html>).

Include the following information:

- Your name and address
- Your email address
- Your phone or fax number
- The publication title and order number:
z/OS MVS Callable Services for HLL
SA23-1377-08
- The topic and page number or URL of the specific information to which your comment relates
- 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 or more of the following actions:

- Visit the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical 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 2, as updated June 2016

The following changes are made for z/OS Version 2 Release 2 (V2R2), as updated June 2016.

New

- “Considerations for Language Environment runtime options” on page 265 has been added.
- New callable services that support the SMF real-time interface have been added in Chapter 22, “SMF real-time interface,” on page 567.
- New REXX API have been added to Chapter 20, “The z/OS JSON parser,” on page 419 section for the HWTJ services.
- New REXX API have been added to Chapter 21, “The z/OS HTTP/HTTPS protocol enabler,” on page 497 section for the HWTH services.
-

Changed

- Updates have been made to the following z/OS client web enablement toolkit information:
 - The description of the **objectHandle** parameter in “HWTJCREN — Create JSON entry” on page 429
 - The descriptions of the **searchStringAddr** and **searchStringLen** parameters in “HWTJSRCH — Search” on page 485
 - “Syntax, linkage, and programming considerations” on page 499
 - The **HWTH_OPT_VERBOSE_OUTPUT** option has been added in “HTTP/HTTPS enabler options and values” on page 552.

Summary of changes for z/OS Version 2 Release 2, as updated March, 2016

The following changes are made for z/OS Version 2 Release 2 (V2R2), as updated March, 2016.

Changed

- Information about the parameters and return codes has been updated in “HWISET — BCPii set SE/HMC-managed attributes” on page 370.

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

The following changes are made for z/OS Version 2 Release 2 (V2R2), as updated December, 2015.

New

- The **forceOption** parameter was added to the HWTHTERM service.
- The following BCPii topics were added: “Setting up an environment to run BCPii TSO/E REXX execs” on page 268 and “Host command environment considerations” on page 268.

Changed

- The description of the HWTHRQST service was updated.
- The description of the **optionValueAddr** parameter for the HWTHSET service was updated.
- The text was updated in “Response body callback routine” on page 563.

Summary of changes for z/OS Version 2 Release 2

The following changes have been made to this publication for z/OS Version 2 Release 2 (V2R2).

New

- Information about BCPii's use of SMF record type 106 has been added in “BCPii configuration considerations” on page 264, “SMF recording in BCPii” on page 270, “HWICMD — Issue a BCPii hardware management command” on page 282, and “HWISET — BCPii set SE/HMC-managed attributes” on page 370.
- New constants, HWI_LAST_USED_LOADADDR and HWI_LAST_USED_LOADPARAM for the **QueryParm_Ptr (QueryParm.)** parameter have been added in “HWIQUERY — BCPii retrieval of SE/HMC-managed attributes” on page 341 and “HWIQUERY and HWISET” on page 585.
- A new **RmrEntriesExact** option has been added in “FPZ4RZV — Rendezvous compression service” on page 208.
- A new topic under BCPii configuration considerations has been added in “Dynamic modification of CPC names” on page 265.
- A new topic, “Setting up access to the HWIREXX helper program” on page 268 has been added, as well as a new return code 112 in “Return codes from the HWIREXX service” on page 274.
- Part 9, “z/OS client web enablement toolkit,” on page 417 has been added, including several new callable services in Chapter 20, “The z/OS JSON parser,” on page 419 and Chapter 21, “The z/OS HTTP/HTTPS protocol enabler,” on page 497.

Changed

- The descriptions of return codes 0-63 (0-99) and D0 (208) have been changed in Appendix A, “BCPii communication error reason codes,” on page 583.
- The statement about the number of TSO/E address spaces the CEA TSO/E address space manager can create on the system or on behalf of a single user has been changed in “Working with TSO/E address spaces started by CEA” on page 145.
- The description of reason code 1004 has been changed in “Reason codes” on page 163.
- The description of diagnostic code 0A has been changed in “Diagnostic codes” on page 167.

Summary of changes for z/OS Version 2 Release 1, as updated September, 2014

The following changes are made for z/OS Version 2 Release 1 (V2R1), as updated September, 2014.

Changed

Changes have been made to the descriptions of the FPZ4ABC and FPZ4RMR compression services.

Changes have been made to some parameter field descriptions for the FPZ4PRB, FPZ4RMR and FPZ4RZV compression services.

Summary of changes for z/OS Version 2 Release 1, as updated March, 2014

The following changes are made for z/OS Version 2 Release 1 (V2R1), as updated March, 2014.

New

New option fields are added in the FPZ4RZV and FPZ4PRB compression services.

Note: For more information on the zEDC compression enhancements, see *z/OS DFSMS Using the New Functions*.

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*

Part 1. Window services

Chapter 1. Introduction to window services

Window services allow HLL programs to:

- Read or update an existing permanent data object
- Create and save a new permanent data object
- Create and use a temporary data object

Window services enable your program to access data objects without your program performing any input or output (I/O) operations. All your program needs to do is issue a CALL to the appropriate service program. The service program performs any I/O operations that are required to make the data object available to your program. When you want to update or save a data object, window services again perform any required I/O operations.

Permanent data objects

A permanent data object is a virtual storage access method (VSAM) linear data set that resides on DASD. (This type of data set is also called a data-in-virtual object.) You can read data from an existing permanent object and also update the content of the object. You can create a new permanent object and when you are finished, save it on DASD. Because you can save this type of object on DASD, window services calls it a permanent object. Window services can handle very large permanent objects that contain as many as 4 gigabytes (four billion bytes).

Note: Installations whose FORTRAN programs used data-in-virtual objects prior to MVS/SP 3.1.0 had to write an assembler language interface program to allow the FORTRAN program to invoke the data-in-virtual program. Window services eliminates the need for this interface program.

Temporary data objects

A temporary data object is an area of expanded storage that window services provides for your program. You can use this storage to hold temporary data, such as intermediate results of a computation, instead of using a DASD workfile. Or you might use the storage area as a temporary buffer for data that your program generates or obtains from some other source. When you finish using the storage area, window services deletes it. Because you cannot save the storage area, window services calls it a temporary object. Window services can handle very large temporary objects that contain as many as 16 terabytes (16 trillion bytes).

Structure of a data object

Think of a data object as a contiguous string of bytes organized into blocks, each 4096 bytes long. The first block contains bytes 0 to 4095 of the object, the second block contains bytes 4096 to 8191, and so forth.

Your program references data in the object by identifying the block or blocks that contain the desired data. Window services makes the blocks available to your program by mapping a window in your program storage to the blocks. A window is a storage area that your program provides and makes known to window services. Mapping the window to the blocks means that window services makes the data from those blocks available in the window when you reference the data.

You can map a window to all or part of a data object depending on the size of the object and the size of the window. You can examine or change data that is in the window by using the same instructions that you use to examine or change any other data in your program storage.

The following figure shows the structure of a data object and shows a window mapped to two of the object's blocks.

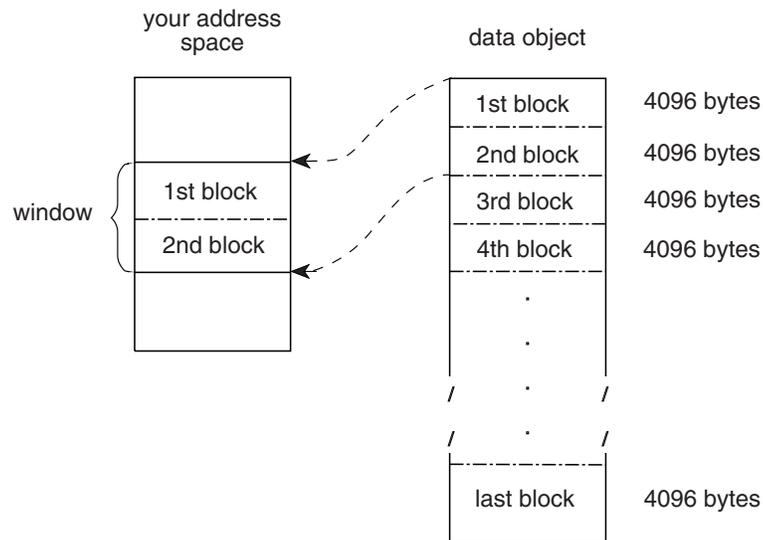


Figure 1. Structure of a Data Object

What does window services provide?

Window services allows you to view and manipulate data objects in a number of ways. You can have access to one or more data objects at the same time. You can also define multiple windows for a given data object. You can then view a different part of the object through each window. Before you can access any data object, you must request access from window services.

When you request access to a permanent data object, you must indicate whether you want a scroll area. A scroll area is an area of expanded storage that window services obtains and maps to the permanent data object. You can think of the permanent object as being available in the scroll area. When you request a view of the object, window services maps the window to the scroll area. If you do not request a scroll area, window services maps the window directly to the object on DASD.

A scroll area enables you to save interim changes to a permanent object without changing the object on DASD. Also, when your program accesses a permanent object through a scroll area, your program might attain better performance than it would if the object were accessed directly on DASD.

When you request a temporary object, window services provides an area of expanded storage. This area of expanded storage is the temporary data object. When you request a view of the object, window services maps the window to the temporary object. Window services initializes a temporary object to binary zeroes.

Note:

1. Window services does not transfer data from the object on DASD, from the scroll area, or from the temporary object until your program references the data. Then window services transfers those blocks.
2. The expanded storage that window services uses for a scroll area or for a temporary object is called a hiperspace. A hiperspace is a range of contiguous virtual storage addresses that a program can indirectly access through a window in the program's virtual storage. Window services uses as many hiperspaces as needed to contain the data object.

The ways that window services can map an object

Window services can map a data object a number of ways. The following examples show how window services can:

- Map a permanent object that has no scroll area
- Map a permanent object that has a scroll area
- Map a temporary object
- Map an object to multiple windows
- Map multiple objects

Example 1 — Mapping a permanent object that has no scroll area

If a permanent object has no scroll area, window services maps the object from DASD directly to your window. In this example, your window provides a view of the first and second blocks of an object.

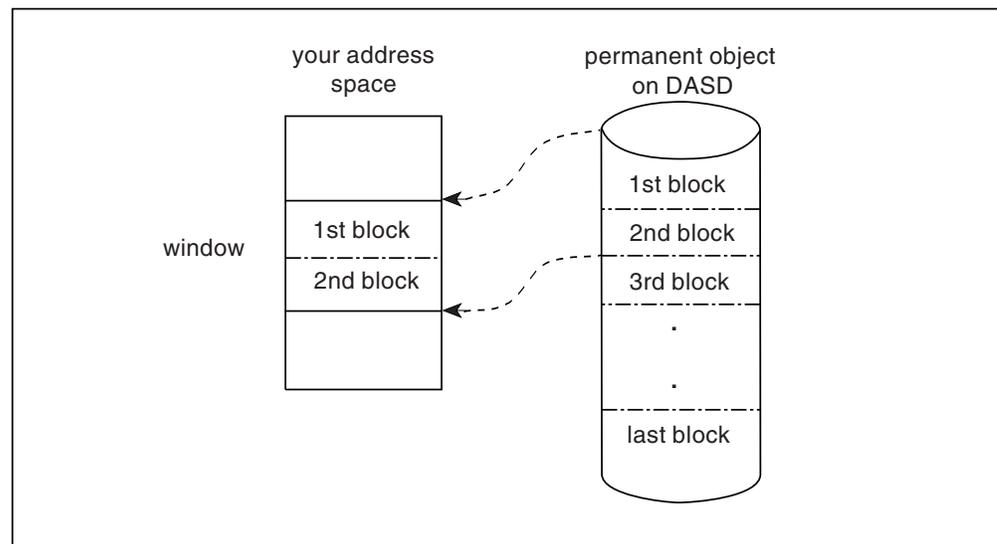


Figure 2. Mapping a Permanent Object That Has No Scroll Area

Example 2 — Mapping a permanent object that has a scroll area

If the object has a scroll area, window services maps the object from DASD to the scroll area. Window services then maps the blocks that you wish to view from the scroll area to your window. In this example, your window provides a view of the third and fourth blocks of an object.

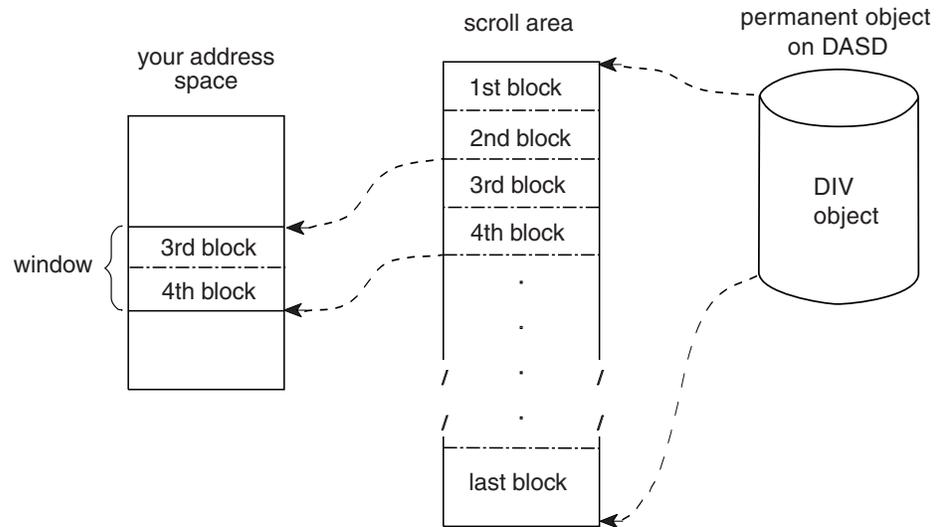


Figure 3. Mapping a Permanent Object That Has a Scroll Area

Example 3 — Mapping a temporary object

Window services uses a hiperspace as a temporary object. In this example, your window provides a view of the first and second blocks of a temporary object.

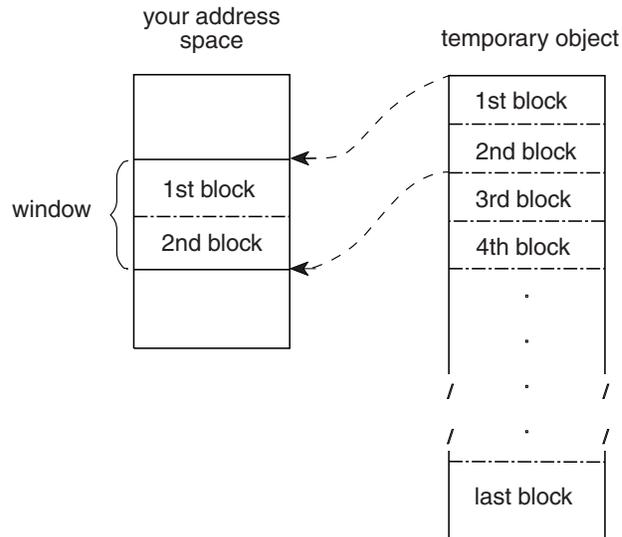


Figure 4. Mapping a Temporary Object

Example 4 — Mapping multiple Windows to an object

Window services can map multiple windows to the same object. In this example, one window provides a view of the second and third blocks of an object, and a second window provides a view of the last block.

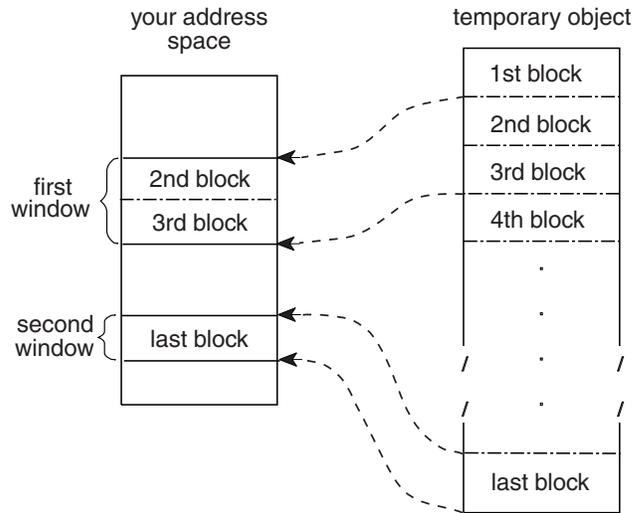


Figure 5. Mapping an Object to Multiple Windows

Example 5 — Mapping multiple objects

Window services can map windows in the same address space to multiple objects. The objects can be temporary objects, permanent objects, or a combination of temporary and permanent objects. In this example, one window provides a view of the second block of a temporary object, and a second window provides a view of the fourth and fifth blocks of a permanent object.

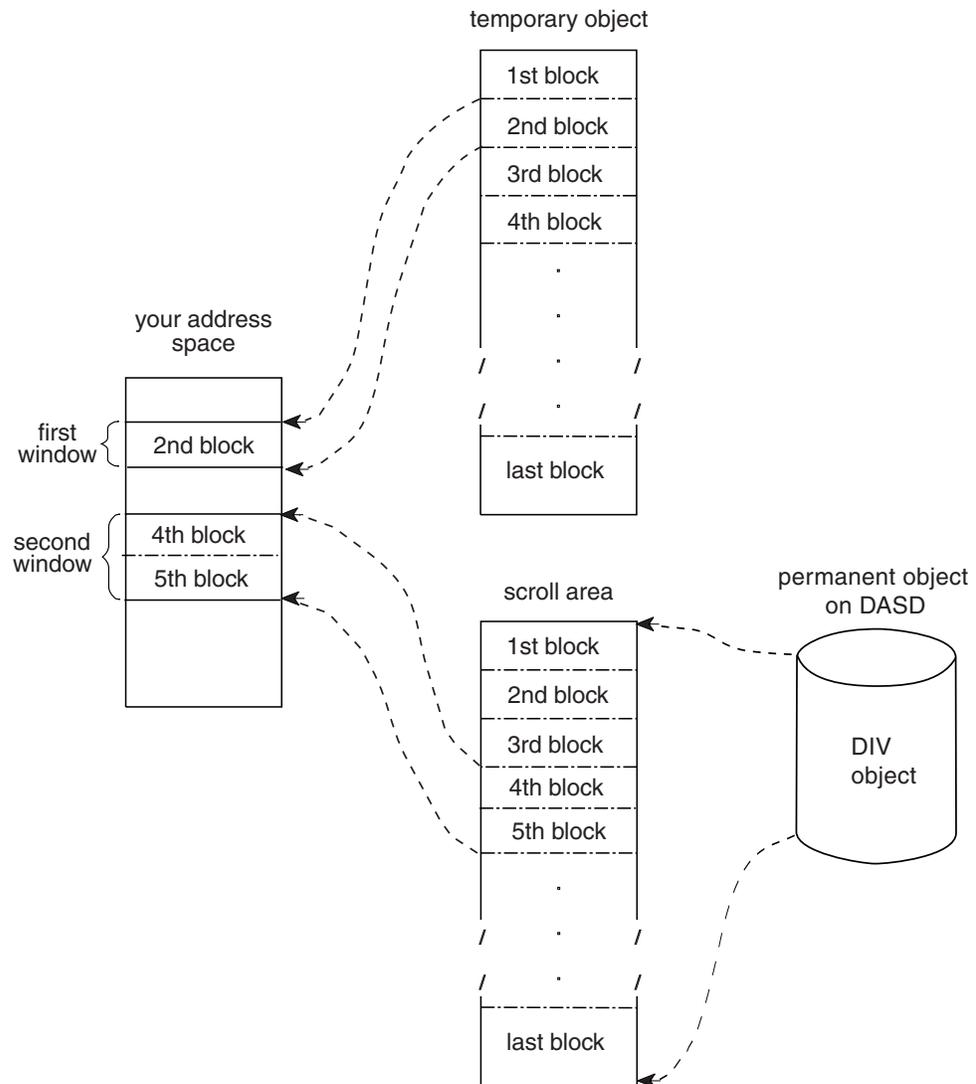


Figure 6. Mapping Multiple Objects

Access to permanent data objects

When you have access to a permanent data object, you can:

- **View the object through one or more windows** — Depending on the object size and the window size, a single window can view all or part of a permanent object. If you define multiple windows, each window can view a different part of the object. For example, one window might view the first block of the permanent object and another window might view the second block. You can also have several windows view the same part of the object or have views in multiple windows overlap. For example, one window might view the first and second blocks of a data object while another window views the second and third blocks.
- **Change data that appears in a window** — You can examine or change data that is in a window by using the same instructions you use to examine or change any other data in your program's storage. These changes do not alter the object on DASD or in the scroll area.

- **Save interim changes in a scroll area** — After changing data in a window, you can have window services save the changed blocks in a scroll area, if you have requested one. Window services replaces blocks in the scroll area with corresponding changed blocks from the window. Saving changes in the scroll area does not alter the object on DASD or alter data in the window.
- **Refresh a window or the scroll area** — After you change data in a window or save changes in the scroll area, you may discover that you no longer need those changes. In that case, you can have window services refresh the changed data. To refresh the window or the scroll area, window services replaces changed data with data from the object as it appears on DASD.
- **Replace the view in a window** — After you finish using data that is in a window, you can have window services replace the view in the window with a different view of the object. For example, if you are viewing the third, fourth, and fifth blocks of an object and are finished with those blocks, you might have window services replace that view with a view of the sixth, seventh, and eighth blocks.
- **Update the object on DASD** — If you have changes available in a window or in the scroll area, you can save the changes on DASD. Window services replaces blocks on DASD with corresponding changed blocks from the window and the scroll area. Updating an object on DASD does not alter data in the window or in the scroll area.

Access to temporary data objects

When you have access to a temporary data object, you can:

- **View the object through one or more windows** — Depending on the object size and the window size, a single window can view all or part of a temporary object. If you define multiple windows, each window can view a different part of the object. For example, one window might view the first block of the temporary object and another window might view the second block. Unlike a permanent object, however, you cannot define multiple windows that have overlapping views of a temporary object.
- **Change data that appears in a window** — This function is the same for a temporary object as it is for a permanent object: you can examine or change data that is in a window by using the same instructions you use to examine or change any other data in your address space.
- **Update the temporary object** — After you have changed data in a window, you can have window services update the object with those changes. Window services replaces blocks in the object with corresponding changed blocks from the window. The data in the window remains as it was.
- **Refresh a window or the object** — After you change data in a window or save changes in the object, you may discover that you no longer need those changes. In that case, you can have window services refresh the changed data. To refresh the window or the object, window services replaces changed data with binary zeroes.
- **Replace the view in a window** — After you finish using data that is in a window, you can have window services replace the view in the window with a different view of the object. For example, if you are viewing the third, fourth, and fifth blocks of an object and are finished with those blocks, you might have window services replace that view with a view of the sixth, seventh, and eighth blocks.

Chapter 2. Using window services

To use, create, or update a data object, you call a series of programs that window services provides. These programs enable you to:

- Access an existing object, create and save a new permanent object, or create a temporary object
- Obtain a scroll area where you can make interim changes to a permanent object
- Define windows and establish views of an object in those windows
- Change or terminate the view in a window
- Update a scroll area or a temporary object with changes you have made in a window
- Refresh changes that you no longer need in a window or a scroll area
- Update a permanent object on DASD with changes that are in a window or a scroll area
- Terminate access to an object

The window services programs that you call and the sequence in which you call them depends on your use of the data object.

The first step in using any data object is to gain access to the object. To gain access, call CSRIDAC. The object can be an existing permanent object, or a new permanent or temporary object you want to create. For a permanent object, you can request an optional scroll area. A scroll area enables you to make interim changes to an object's data without affecting the data on DASD. When CSRIDAC grants access, it provides an object identifier that identifies the object. Use that identifier to identify the object when you request other services from window services.

After obtaining access to an object, define one or more windows and establish views of the object in those windows. To establish a view of an object, tell window services which blocks you want to view and in which windows. You can view multiple objects and multiple parts of each object at the same time. To define windows and establish views, call CSRVIEW or CSREVV. After establishing a view, you can examine or change data that is in the window using the same instructions you use to examine or change other data in your program's storage.

After making changes to the part of an object that is in a window, you will probably want to save those changes. How you save changes depends on whether the object is permanent, is temporary, or has a scroll area.

If the object is permanent and has a scroll area, you can save changes in the scroll area without affecting the object on DASD. Later, you can update the object on DASD with changes saved in the scroll area. If the object is permanent and has no scroll area, you can update it on DASD with changes that are in a window. If the object is temporary, you can update it with changes that are in a window. To update an object on DASD, call CSRSAVE. To update a temporary object or a scroll area, call CSRSCOT.

After making changes in a window and possibly saving them in a scroll area or using them to update a temporary object, you might decide that you no longer need those changes. In this case, you can refresh the changed blocks. After refreshing a block of a permanent object or a scroll area to which a window is

mapped, the refreshed block contains the same data that the corresponding block contains on DASD. After refreshing a block of a temporary object to which a window is mapped, the block contains binary zeroes. To refresh a changed block, call CSRREFR.

After finishing with a view in a window, you can use the same window to view a different part of the object or to view a different object. Before changing the view in a window, you must terminate the current view. If you plan to view a different part of the same object, terminate the current view by calling CSRVIEW. If you plan to view a different object or will not reuse the window, you can terminate the view by calling CSRIDAC.

When you finish using a data object, terminate access to the object by calling CSRIDAC.

The following restrictions apply to using window services:

1. When you attach a new task, you cannot pass ownership of a mapped virtual storage window to the new task. That is, you cannot use the ATTACH or ATTACHX keywords GSPV and GSPL to pass the mapped virtual storage.
2. While your program is in cross-memory mode, your program cannot invoke data-in-virtual services; however, your program can reference and update data in a mapped virtual storage window.
3. The task that obtains the ID (through DIV IDENTIFY) is the only one that can issue other DIV services for that ID.
4. When you identify a data-in-virtual object using the IDENTIFY service, you cannot request a checkpoint until you invoke the corresponding UNIDENTIFY service.

This topic explains how to do the previously described functions and contains the following subtopics:

- “Obtaining access to a data object”
- “Defining a view of a data object” on page 14
- “Defining multiple views of an object” on page 17
- “Saving interim changes to a permanent data object” on page 18
- “Updating a temporary data object” on page 18
- “Refreshing changed data” on page 19
- “Updating a permanent object on DASD” on page 20
- “Changing a view in a window” on page 20
- “Terminating access to a data object” on page 22
- “Handling return codes and abnormal terminations” on page 22.

Obtaining access to a data object

To obtain access to a permanent or temporary data object, call CSRIDAC. Indicate that you want to access an object by specifying BEGIN as the value for *op_type*. For a description of the CSRIDAC parameters and return codes, see “CSRIDAC — Request or terminate access to a data object” on page 27.

Identifying the object

You must identify the data object you wish to access. How you identify the object depends on whether the object is permanent or temporary.

Permanent object

For a permanent object, *object_name* and *object_type* work together. For *object_name* you have a choice: specify either the data set name of the object or the DDNAME to which the object is allocated. The *object_type* parameter must then indicate whether *object_name* is a DDNAME or a data set name:

- If *object_name* is a DDNAME, specify DDNAME as the value for *object_type*.
- If *object_name* is a data set name, specify DSNNAME as the value for *object_type*.

If you specify DSNNAME for *object_type*, indicate whether the object already exists or whether window services is to create it:

- If the object already exists, specify OLD as the value for *object_state*.
- If window services is to create the object, specify NEW as the value for *object_state*.

Note: Requirement for NEW objects: If you specify NEW as the value for *object_state*, your system must include MVS/Data Facility Product. (MVS/DFP) 3.1.0 and SMS must be active.

Temporary object

To identify a temporary object, specify TEMPSPACE as the value for *object_type*. Window services assumes that a temporary object is new and ignores the value that you specify for *object_state*.

Specifying the object's size

If the object is permanent and new or is temporary, you must tell window services the size of the object. You specify object size through the *object_size* parameter. The size specified becomes the maximum size that window services will allow for that object. You express the size as a number of 4096-byte blocks. If the number of bytes in the object is not an exact multiple of 4096, round *object_size* to the next whole number. For example:

- If the object size is to be less than 4097 bytes, specify 1.
- If the object size is 5000 bytes, specify 2.
- If the object size is 410,000 bytes, specify 101.

Specifying the type of access

For an existing (OLD) permanent object, you must specify how you intend to access the object. You specify your intentions through the *access_mode* parameter:

- If you intend to only read the object, specify READ for *access_mode*.
- If you intend to update the object, specify UPDATE for *access_mode*.

For a new permanent object and for a temporary object, window services assumes you will update the object and ignores the value you specify for *access_mode*.

Obtaining a scroll area

A scroll area is storage that window services provides for your use. This storage is outside your program's storage area and is accessible only through window services.

For a permanent object, a scroll area is optional. A scroll area allows you to make interim changes to a permanent object without altering the object on DASD. Later,

if you want, you can update the object on DASD with the interim changes. A scroll area might also improve performance when your program accesses a permanent object.

For a temporary object, the scroll area is the object. Therefore, for a temporary object, a scroll area is required.

To indicate whether you want a scroll area, provide the appropriate value for *scroll_area*:

- To request a scroll area, supply a value of YES. YES is required for a temporary object.
- To indicate you do not want a scroll area, supply a value of NO.

Defining a view of a data object

To view all or part of a data object, you must provide window services with information about the object and how you want to view it. You must provide window services with the following information:

- The object identifier
- Where the window is in your address space
- Window disposition — that is, whether window services is to initialize the window the first time you reference data in the window
- Whether you intend to reference blocks of data sequentially or randomly
- The blocks of data that you want to view
- Whether you want to extend the size of the object

To define a view of a data object, call CSRVIEW or CSREVIEW. Whether you use CSRVIEW or CSREVIEW depends on how you plan to reference the data. “Defining the expected reference pattern” on page 15 describes the differences between the two services. Specify BEGIN on CSRVIEW or CSREVIEW as the type of operation. For descriptions of the CALL syntax and return codes from CSRVIEW or CSREVIEW, see “CSRVIEW — View an object” on page 39 or “CSREVIEW — View an object and sequentially access it” on page 23.

Identifying the data object

To identify the object you want to view, specify the object identifier as the value for *object_id*. Use the same value CSRIDAC returned in *object_id* when you requested access to the object.

Identifying a window

You must identify the window through which you will view the object. The window is a virtual storage area in your address space. You are responsible for obtaining the storage, which must meet the following requirements:

- The storage must not be page fixed.
- Pages in the window must not be page loaded (must not be loaded by the PGLOAD macro).
- The storage must start on a 4K boundary and must be a multiple of 4096 bytes in length.

To identify the window, use the *window_name* parameter. The value supplied for *window_name* must be the symbolic name you assigned to the window storage area in your program.

Defining a window in this way provides one window through which you can view the object. To define multiple windows that provide simultaneous views of different parts of the object, see “Defining multiple views of an object” on page 17.

Defining the disposition of a window’s contents

You must specify whether window services is to replace or retain the window contents. You do this by selecting either the replace or retain option. This option determines how window services handles the data that is in the window the first time you reference the data. You select the option by supplying a value of REPLACE or RETAIN for *disposition*.

Replace option

If you specify the replace option, the first time you reference a block to which a window is mapped, window services replaces the data in the window with corresponding data from the object. For example, assume you have requested a view of the first block of a permanent object and have specified the replace option. The first time you reference the window, window services replaces the data in the window with the first 4096 bytes (the first block) from the object.

If you have selected the replace option and then call CSRSAVE to update a permanent object, or call CSRSCOT to update a scroll area, or call CSRSCOT to update a temporary object, window services updates only the specified blocks that have changed and to which a window is mapped.

Select the replace option when you want to examine, use, or change data that is currently in an object.

Retain option

If you select the retain option, window services retains data that is in the window. When you reference a block in the window the first time, the block contains the same data it contained before the reference.

When you select the retain option, window services considers all of the data in the window as changed. Therefore, if you call CSRSCOT to update a scroll area or a temporary object, or call CSRSAVE to update a permanent object, window services updates all of the specified blocks to which a window or scroll area are mapped.

Select the retain option when you want to replace data in an object without regard for the data that it currently contains. You also use the retain option when you want to initialize a new object.

Defining the expected reference pattern

You must tell window services whether you intend to reference the blocks of an object sequentially or randomly. An intention to access randomly tells window services to bring one block (4096 bytes) of data into the window at a time. An intention to access sequentially tells window services to read more than one block into your window at one time. The performance gain is in having blocks of data already in central storage at the time the program needs to reference them. You specify the intent on either CSRVIEW or CSREVVW, two services that differ on how to specify sequential access.

- CSRVIEW allows you a choice between random or sequential access.

If you specify **random**, when you reference data that is not in your window, window services brings in one block — the one that contains the data your program references.

If you specify **sequential**, when you reference data that is not in your window, window services transfers up to 16 blocks — the one that contains the data your program requests, plus the next 15 consecutive blocks. The number of consecutive blocks varies, depending on the size of the window and availability of central storage. Use CSRVIEW if one of the following is true:

- You are going to access randomly.
- You are going to access sequentially, and you are satisfied with a maximum of 16 blocks coming into the window at a time.
- CSREVV is for sequential access only. It allows you to specify the maximum number of consecutive blocks that window services brings into the window at one time. The number ranges from one block through 256 blocks. Use CSREVV if you want fewer than 16 blocks or more than 16 blocks at one time. Programs that benefit from having more than 16 blocks come into a window at one time reference data areas that are greater than one megabyte.

To specify the reference pattern on CSRVIEW, supply a value of SEQ or RANDOM for *usage*.

To specify the reference pattern on CSREVV, supply a number from 0 through 255 for *pfcount*. *pfcount* represents the number of blocks window services will bring into the window, in addition to the one that it always brings in.

Note that window services brings in multiple pages differently depending on whether your object is permanent or temporary and whether the system has had to move pages of your data from central storage to make those pages of central available for other programs. The rule is that SEQ on CSRVIEW and *pfcount* on CSREVV apply to:

- A permanent object when movement is from the object on DASD to central storage
- A temporary object when your program has scrolled the data out and references it again

SEQ and *pfcount* do not apply after the system has had to move data (either changed or unchanged) to auxiliary or expanded storage, and your program again references it, requiring the system to bring the data back into central storage.

End the view, whether established with CSRVIEW or CSREVV, with CSRVIEW END.

Identifying the blocks you want to view

To identify the blocks of data you want to view, use *offset* and *span*. The values you assign to *offset* and *span*, together, define a contiguous string of blocks that you want to view:

- The value assigned to *offset* specifies the relative block at which to start the view. An offset of 0 means the first block; an offset of 1 means the second block; an offset of 2 means the third block, and so forth.
- The value assigned to *span* specifies the number of blocks to view. A span of 1 means one block; a span of 2 means two blocks, and so forth. A span of 0 has special meaning: it means the view is to start at the specified offset and extend until the currently defined end of the object.

The following table shows examples of several *offset* and *span* combinations and the resulting view in the window.

Offset	Span	Resulting view in the window
0	0	view the entire object
0	1	view the first block only
1	0	view the second block through the last block
1	1	view the second block only
2	2	view the third and fourth blocks only

Extending the size of a data object

You can use *offset* and *span* to extend the size of an object up to the previously defined maximum size for the object. You can extend the size of either permanent objects or temporary objects. For objects created through CSRIDAC, the value assigned to *object_size* defines the maximum allowable size. When you call CSRIDAC to gain access to an object, CSRIDAC returns a value in *high_offset* that defines the current size of the object.

For example, assume you have access to a permanent object whose maximum allowable size is four 4096-byte blocks. The object is currently two blocks long. If you define a window and specify an offset of 1 and a span of 2, the window contains a view of the second block and a view of a third block, which does not yet exist in the permanent object. When you reference the window, the content of the second block, as seen in the window, depends on the disposition you selected, replace or retain. The third block, as seen in the window, initially contains binary zeroes. If you later call CSRSAVE to update the permanent object with changes from the window, window services extends the size of the permanent object to three blocks by appending the new block of data to the object.

Defining multiple views of an object

You might need to view different parts of an object at the same time. For a permanent object, you can define windows that have non-overlapping views as well as windows that have overlapping views. For a temporary object, you can define windows that have only non-overlapping views.

- A non-overlapping view means that no two windows view the same block of the object. For example, a view is non-overlapping when one window views the first and second blocks of an object and another window views the ninth and tenth blocks of the same object. Neither window views a common block.
- An overlapping view means that two or more windows view the same block of the object. For example, the view overlaps when the second window in the previous example views the second and third blocks. Both windows view a common block, the second block.

Non-overlapping views

To define multiple windows that have a non-overlapping view, call CSRIDAC once to obtain the object identifier. Then call CSRVIEW or CSREVIEW once to define each window. On each call, specify the value BEGIN for *operation_type*, the same object identifier for *object_id*, and a different value for *window_name*. Define each window's view by specifying values for *offset* and *span* that create windows with non-overlapping views.

Overlapping views

To define multiple windows that have an overlapping view of a permanent object, define each window as though it were viewing a different object. That is, define each window under a different object identifier. To obtain the object identifiers, call

CSRIDAC once for each identifier you need. Only one of the calls to CSRIDAC can specify an access mode of UPDATE. Other calls to CSRIDAC must specify an access mode of READ.

After calling CSRIDAC, call CSRVIEW or CSREVIEW once to define each window. On each call, specify the value BEGIN for the operation type, a different object identifier for *object_id*, and a different value for *window_name*. Define each window's view by specifying values for *offset* and *span* that create windows with the required overlapping views.

Saving interim changes to a permanent data object

Window services allows you to save interim changes you make to a permanent object. You must have previously requested a scroll area for the object, however. You request a scroll area when you call CSRIDAC to gain access to the object. Window services saves changes by replacing blocks in the scroll area with corresponding changed blocks from a window. Saving changes in the scroll area does not alter the object on DASD.

After you have a view of the object and have made changes in the window, you can save those changes in the scroll area. To save changes in the scroll area, call CSRSCOT. For a description of the CSRSCOT parameters and return codes, see "CSRSCOT — Save object changes in a scroll area" on page 36.

To identify the object, you must supply an object identifier for *object_id*. The value supplied for *object_id* must be the same value CSRIDAC returned in *object_id* when you requested access to the object.

To identify the blocks in the object that you want to update, use *offset* and *span*. The values assigned to *offset* and *span*, together, define a contiguous string of blocks in the object:

- The value assigned to *offset* specifies the relative block at which to start. An offset of 0 means the first block; an offset of 1 means the second block; an offset of 2 means the third block, and so forth.
- The value assigned to *span* specifies the number of blocks to save. A span of 1 means one block; a span of 2 means two blocks, and so forth. A span of 0 has special meaning: it requests that window services save all changed blocks to which a window is mapped.

Window services replaces each block within the range specified by *offset* and *span* providing the block has changed and a window is mapped to the block.

Updating a temporary data object

After making changes in a window to a temporary object, you can update the object with those changes. You must identify the object and must specify the range of blocks that you want to update. To be updated, a block must be mapped to a window and must contain changes in the window. Window services replaces each block within the specified range with the corresponding changed block from a window.

To update a temporary object, call CSRSCOT. For a description of the CSRSCOT parameters and return codes, see "CSRSCOT — Save object changes in a scroll area" on page 36.

To identify the object, you must supply an object identifier for *object_id*. The value you supply for *object_id* must be the same value CSRIDAC returned in *object_id* when you requested access to the object.

To identify the blocks in the object that you want to update, use *offset* and *span*. The values assigned to *offset* and *span*, together, define a contiguous string of blocks in the object:

- The value assigned to *offset* specifies the relative block at which to start. An offset of 0 means the first block; an offset of 1 means the second block; an offset of 2 means the third block, and so forth.
- The value assigned to *span* specifies the number of blocks to save. A span of 1 means one block; a span of 2 means two blocks, and so forth. A span of 0 has special meaning: it requests that window services update all changed blocks to which a window is mapped.

Window services replaces each block within the range specified by *offset* and *span* providing the block has changed and a window is mapped to the block.

Refreshing changed data

You can refresh blocks that are mapped to either a temporary object or to a permanent object. You must identify the object and specify the range of blocks you want to refresh. When you refresh blocks mapped to a temporary object, window services replaces, with binary zeros, all changed blocks that are mapped to the window. When you refresh blocks mapped to a permanent object, window services replaces specified changed blocks in a window or in the scroll area with corresponding blocks from the object on DASD.

To refresh an object, call CSRREFR. For a description of CSRREFR parameters and return codes, see “CSRREFR — Refresh an object” on page 31.

To identify the object, you must supply an object identifier for *object_id*. The value supplied for *object_id* must be the same value CSRIDAC returned in *object_id* when you requested access to the object.

To identify the blocks of the object that you want to refresh, use *offset* and *span*. The values assigned to *offset* and *span*, together, define a contiguous string of blocks in the object:

- The value assigned to *offset* specifies the relative block at which to start. An offset of 0 means the first block; an offset of 1 means the second block; an offset of 2 means the third block, and so forth.
- The value assigned to *span* specifies the number of blocks to save. A span of 1 means one block; a span of 2 means two blocks, and so forth. A span of 0 has special meaning: it requests that window services refresh all changed blocks to which a window is mapped, or that have been saved in a scroll area.

Window services refreshes each block within the range specified by *offset* and *span* providing the block has changed and a window or a scroll area is mapped to the block. At the completion of the refresh operation, blocks from a permanent object that have been refreshed appear the same as the corresponding blocks on DASD. Refreshed blocks from a temporary object contain binary zeroes.

Updating a permanent object on DASD

You can update a permanent object on DASD with changes that appear in a window or in the object's scroll area. You must identify the object and specify the range of blocks that you want to update.

To update an object, call CSRSAVE. For a description of the CSRSAVE parameters and return codes, see "CSRSAVE — Save changes made to a permanent object" on page 34.

To identify the object, you must supply an object identifier for *object_id*. The value you provide for *object_id* must be the same value CSRIDAC returned when you requested access to the object.

To identify the blocks of the object that you want to update, use *offset* and *span*. The values assigned to *offset* and *span*, together, define a contiguous string of blocks in the object:

- The value assigned to *offset* specifies the relative block at which to start. An offset of 0 means the first block; an offset of 1 means the second block; an offset of 2 means the third block, and so forth.
- The value assigned to *span* specifies the number of blocks to save. A span of 1 means one block; a span of 2 means two blocks, and so forth. A span of 0 has special meaning: it requests that window services update all changed blocks to which a window is mapped, or have been saved in the scroll area.

When there is a scroll area

When the object has a scroll area, window services first updates blocks in the scroll area with corresponding blocks from windows. To be updated, a scroll area block must be within the specified range, a window must be mapped to the block, and the window must contain changes. Window services next updates blocks on DASD with corresponding blocks from the scroll area. To be updated, a DASD block must be within the specified range and have changes in the scroll area. Blocks in the window remain unchanged.

When there is no scroll area

When there is no scroll area, window services updates blocks of the object on DASD with corresponding blocks from a window. To be updated, a DASD block must be within the specified range, mapped to a window, and have changes in the window. Blocks in the window remain unchanged.

Changing a view in a window

To change the view in a window so you can view a different part of the same object or view a different object, you must first terminate the current view. To terminate the view, whether the view was mapped by CSRVIEW or CSREVV, call CSRVIEW and supply a value of END for *operation_type*. You must also identify the object, identify the window, identify the blocks you are currently viewing, and specify a disposition for the data that is in the window. For a description of CSRVIEW parameters and return codes, see "CSRVIEW — View an object" on page 39.

To identify the object, supply an object identifier for *object_id*. The value supplied for *object_id* must be the value you supplied when you established the view.

To identify the window, supply the window name for *window_name*. The value supplied for *window_name* must be the same value you supplied when you established the view.

To identify the blocks you are currently viewing, supply values for *offset* and *span*. The values you supply must be the same values you supplied for *offset* and *span* when you established the view.

To specify a disposition for the data you are currently viewing, supply a value for *disposition*. The value determines what data will be in the window after the CALL to CSRVIEW completes.

- For a permanent object that has no scroll area:
 - To retain the data that is currently in the window, supply a value of RETAIN for *disposition*.
 - To discard the data that is currently in the window, supply a value of REPLACE for *disposition*. After the operation completes, the window contents are unpredictable.

For example, assume that a window is mapped to one block of a permanent object that has no scroll area. The window contains the character string AAA.....A and the block to which the window is mapped contains BBB.....B. If you specify a value of RETAIN, upon completion of the CALL, the window still contains AAA.....A, and the mapped block contains BBB.....B. If you specify a value of REPLACE, upon completion of the CALL, the window contents are unpredictable and the mapped block still contains BBB.....B.

- For a permanent object that has a scroll area or for a temporary object:
 - To retain the data that is currently in the window, supply a value of RETAIN for *disposition*. CSRVIEW also updates the mapped blocks of the scroll area or temporary object so that they contain the same data as the window.
 - To discard the data that is currently in the window, supply a value of REPLACE for *disposition*. Upon completion of the operation, the window contents are unpredictable.

For example, assume that a window is mapped to one block of a temporary object. The window contains the character string AAA.....A and the block to which the window is mapped contains BBB.....B. If you specify a value of RETAIN, upon completion of the CALL, the window still contains AAA.....A and the mapped block of the object also contains AAA.....A. If you specify a value of REPLACE, upon completion of the CALL, the window contents are unpredictable and the mapped block still contains BBB.....B.

CSRVIEW ignores the values you assign to the other parameters.

When you terminate the view of an object, the type of object that is mapped and the value you specify for *disposition* determine whether CSRVIEW updates the mapped blocks. CSRVIEW updates the mapped blocks of a temporary object or a permanent object's scroll area if you specify a disposition of RETAIN. In all other cases, to update the mapped blocks, call the appropriate service before terminating the view:

- To update a temporary object, or to update the scroll area of a permanent object, call CSRSCOT.
- To update an object on DASD, call CSRSAVE.

Upon successful completion of the CSRVIEW operation, the content of the window depends on the value specified for *disposition*. The window is no longer mapped

to a scroll area or to an object, however. The storage used for the window is available for other use, perhaps to use as a window for a different part of the same object or to use as a window for a different object.

Terminating access to a data object

When you finish using a data object, you must terminate access to the object. When you terminate access, window services returns to the system any virtual storage it obtained for the object: storage for a temporary object or storage for a scroll area. If the object is temporary, window services deletes the object. If the object is permanent and window services dynamically allocated the data set when you requested access to the object, window services dynamically unallocates the data set. Your window is no longer mapped to the object or to a scroll area.

When you terminate access to a permanent object, window services does not update the object on DASD with changes that are in a window or the scroll area. To update the object, call CSRSAVE before terminating access to the object.

To terminate access to an object, call CSRIDAC and supply a value of END for *operation_type*. To identify the object, supply an object identifier for *object_id*. The value you supply for *object_id* must be the same value CSRIDAC returned when you obtained access to the object.

Upon successful completion of the call, the storage used for the window is available for other use, perhaps as a window for viewing a different part of the same object or to use as a window for viewing a different object.

Handling return codes and abnormal terminations

Each time you call a service, your program receives either a return code and reason code or an abend code and a reason code. These codes indicate whether the service completed successfully, encountered an unusual condition, or was unable to complete successfully.

When you receive a return code that indicates a problem or an unusual condition, your program can either attempt to correct the problem or can terminate its execution. Return codes and reason codes are explained in Chapter 3, "Window services," on page 23 with the description of each callable service program.

When an abend occurs, the system passes control to a recovery routine, if you or your installation have provided one. A recovery routine might be able to correct the problem that caused the abend and allow your program to continue execution. If a recovery routine has been provided, it can handle the abend condition the same way it handles other abend conditions. If a recovery routine has not been provided, the system terminates execution of your program. For an explanation of the abend codes, see *z/OS MVS System Codes*.

Chapter 3. Window services

To use window services, you issue CALLs that invoke the appropriate window services program. Each service program performs one or more functions and requires a set of parameters coded in a specific order on the CALL statement.

Depending on the function requested from a service, there might be one or more parameter values that the service ignores. Although a service might ignore a parameter value, you must still code that parameter on the CALL statement. Because the service ignores the parameter value, you can assign the parameter any value that is acceptable for the parameter's data type. If the service uses a particular parameter value, the CALL statement description in this topic defines the allowable values that you can assign to the parameter.

This topic describes the CALL statements that invoke window services. Each description includes a syntax diagram, parameter descriptions, and return code and reason code explanations with recommended actions. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. For examples of how to code the CALL statements, see Chapter 4, "Window services coding examples," on page 45.

This topic contains the following subtopics:

- "CSREVV — View an object and sequentially access it"
- "CSRIDAC — Request or terminate access to a data object" on page 27
- "CSRREFR — Refresh an object" on page 31
- "CSRSAVE — Save changes made to a permanent object" on page 34
- "CSRSCOT — Save object changes in a scroll area" on page 36
- "CSRVIEW — View an object" on page 39

CSREVV — View an object and sequentially access it

Call CSREVV if you reference data in a sequential pattern and you want to:

- Map a window to one or more blocks (4096 bytes) of a data object. If you specified scrolling when you called CSRIDAC to identify the object, CSREVV maps the window to the blocks in the scroll area and maps the scroll area to the object.
- Specify how many blocks window services is to bring into the window each time CSREVV needs more data from the object.

Mapping a data object enables your program to access the data that is viewed through the window the same way it accesses other data in your storage.

The CSREVV and CSRVIEW services differ on how to specify sequential access:

- If you use CSRVIEW and specify sequential, when you reference data that is not in your window, window services reads up to 16 blocks — the one that contains the data your program requests, plus the next 15 consecutive blocks. The number of consecutive blocks varies, depending on the size of the window and the availability of central storage.

CSREVV

- If you use CSREVV, you can specify the number of additional consecutive blocks that window services reads into the window at one time. The number ranges from 0 through 255.

Use CSREVV if your program has sequential access and can benefit from having more than 16 blocks come into a window at one time, or fewer than 16 blocks at one time.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown below. For parameters that CSREVV uses to obtain input values, assign appropriate values. For parameters that CSREVV ignores, assign any value that is valid for the particular parameter's data type.

- To map a window to a data object and begin viewing the object, specify BEGIN and SEQ and assign values, acceptable to CSREVV, to:
 - *object_id*
 - *offset*
 - *span*
 - *window_name*
 - *disposition*
 - *pfcount*
- CSREVV returns values in *return_code* and in *reason_code*. To end the view and unmap the data object, use CSRVIEW END and specify all values, except for *pfcount*, that you specified when you mapped the window.

CALL statement	Parameters
CALL CSREVV	(operation_type ,object_id ,offset ,span ,window_name ,usage ,disposition ,pfcount ,return_code ,reason_code)

operation_type

Specify BEGIN to request that CSREVV map a data object.

,object_id

Specifies the object identifier. Supply the object identifier that CSRIDAC returned when you obtained access to the object.

Define *object_id* as character data of length 8.

,offset

Specifies the offset of the view into the object. Specify the offset in blocks of 4096 bytes.

Define *offset* as integer data of length 4.

,span

Specifies the window size in blocks of 4096 bytes.

Define *span* as integer data of length 4.

,window_name

Specifies the symbolic name you assigned to the window in your address space.

,usage

Specify SEQ to tell CSREVV that the expected pattern of references to data in the object will be sequential.

Define this field as character data of length 6. Pad the string on the right with 1 blank.

,disposition

Defines how CSREVV is to handle data that is in the window when you begin a view. When you specify CSREVV BEGIN and a disposition of:

REPLACE

The first time you reference a block to which the window is mapped, CSREVV replaces the data in the window with the data from the referenced block.

RETAIN

When you reference a block to which the window is mapped, the data in the window remains unchanged. When you call CSRSAVE to save the mapped blocks, CSRSAVE saves all of the mapped blocks because CSRSAVE considers them changed.

Define *disposition* as character data of length 7. If you specify RETAIN, pad the string on the right with 1 blank.

,pfcount

Specifies the number of additional blocks you want window services to bring into the window each time your program references data that is not already in the window. The number you specify is added to the minimum of one block that window services always brings in. That is, if you specify a value of 20, window services brings in a total of 21. The number of additional blocks ranges from zero through 255.

Define *pfcount* as integer data of length 4.

,return_code

When CSREVV completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

Return codes and reason codes are explained under “Return codes and reason codes” on page 26.

,reason_code

When CSREVV completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes are explained under “Return codes and reason codes” on page 26.

Abend codes

CSREVV issues abend code X'019'. For more information, see *z/OS MVS System Codes*.

Return codes and reason codes

When CSREVV returns control to your program, *return_code* contains a return code and *reason_code* contains a reason code. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. Table 1 identifies return code and reason code combinations, tells what each means, and recommends an action that you should take.

A return code of X'4' with a reason code of X'0125' or a return code of X'C' with any reason code means that data-in-virtual encountered a problem or an unexpected condition. Data-in-virtual reason codes, which are two bytes long and right justified, are explained in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. To resolve a data-in-virtual problem, request help from your system programmer.

Table 1. CSREVV Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	Meaning: The operation was successful. Action: Continue normal program execution.
00000004 (4)	xxxx0125 (293)	Meaning: The operation was successful. The service could not retain all the data that was in the scroll area, however. Action: Notify your system programmer.
00000012 (18)	xxxx000A (10)	Meaning: There is another service currently executing with the specified ID. Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	Meaning: An I/O error has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx001A (26)	Meaning: The specified range does not encompass any mapped area of the object. Action: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx001C (28)	Meaning: The object cannot be accessed at the current time. Action: Try running your program at a later time. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0040 (64)	Meaning: The specified MAP range would cause the hiperspace data-in-virtual object to be extended such that the installation data space limits would be exceeded. Action: Change the MAP range you have specified or request your system programmer to increase the installation's data space limits.
0000000C (12)	xxxx0801 (2049)	Meaning: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. Action: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	Meaning: System error — I/O driver failure. Action: Notify your system programmer.

Table 1. CSREVV Return and Reason Codes (continued)

Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx0803 (2051)	Meaning: System error — A necessary page table could not be read into real storage. Action: Notify your system programmer.
0000000C (12)	xxx00804 (2052)	Meaning: System error — Catalog update failed. Action: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	Meaning: System error — I/O error. Action: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	Meaning: System error — I/O from a previous request has not completed. Action: Notify your system programmer.
0000002C (44)	00000004 (4)	Meaning: Window services have not been defined to your system or the link to the service failed. Action: Notify your system programmer.

CSRIDAC — Request or terminate access to a data object

Call CSRIDAC to:

- Request access to a data object
- Terminate access to a data object

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown. For parameters that CSRIDAC uses to obtain input values, assign values that are acceptable to CSRIDAC. For parameters that CSRIDAC ignores, assign any value that is valid for the particular parameter's data type.

The parameter values that CSRIDAC uses depends on whether you are requesting access to an object or terminating access.

- To request access to a data object, specify BEGIN for *operation_type*, and assign values, acceptable to CSRIDAC, to the following parameters:
 - *object_type*
 - *object_name* if the object is permanent
 - *scroll_area*
 - *object_state* if the object is permanent and *object_type* specifies DSNAME
 - *access_mode* if the object exists and is permanent
 - *object_size* if the object is new or temporary
 - *object_size* if the object is new or temporary

CSRIDAC ignores other parameter values. CSRIDAC returns values in *object_id*, *high_offset*, *return_code*, and *reason_code*.

- To terminate access to a data object, specify END for *operation_type*, and assign a value, acceptable to CSRIDAC, to *object_id*. CSRIDAC ignores other parameter values. CSRIDAC returns values in *return_code* and *reason_code*.

CALL statement	Parameters
CALL CSRIDAC	(operation_type ,object_type ,object_name ,scroll_area ,object_state ,access_mode ,object_size ,object_id ,high_offset ,return_code ,reason_code)

operation_type

Specifies the type of operation the service is to perform:

- To request access to an object, specify BEGIN.
- To terminate access to an object, specify END. If the object is temporary, CSRIDAC deletes it.

Define *operation_type* as character data of length 5. If you specify END, pad the string on the right with 1 or 2 blanks.

,object_type

Specifies the type of object. The types are:

DDNAME

The object is an existing (OLD) VSAM linear data set allocated to the file whose DDNAME is specified by *object_name*.

DSNAME

The object is the linear VSAM data set whose name is specified by *object_name*. The data set may already exist or may be a new data set that you want window services to create.

TEMPSPACE

The object is a temporary data object. Window services deletes the object when your program calls CSRIDAC and *operation_type* equals END.

If *operation_type* is BEGIN, you must supply a value.

Define this parameter as character data of length 9. If you specify either DDNAME or DSNAME, pad the string on the right with 1 to 3 blanks.

,object_name

Specifies the data set name of a permanent object or the DDNAME of a data definition (DD) statement that defines a permanent object.

- If *object_type* is DDNAME, *object_name* must contain the name of a DD statement.
- If *object_type* is DSNAME, *object_name* must contain the data set name of the permanent object.

If *operation_type* is BEGIN and *object_type* is DDNAME or DSNAME, you must supply a value for *object_name*.

Define *object_name* as character data of length 1 to 45. If *object_name* contains fewer than 45 characters, pad the name on the right with a blank.

,scroll_area

Specifies whether window services is to create a scroll area for the data object.

YES Create a scroll area.

NO Do not create a scroll area.

If *operation_type* is BEGIN and *object_type* is TEMPSPACE, specify YES.

Define *scroll_area* as character data of length 3. If you specify NO, pad the string on the right with a blank.

,object_state

Specifies the state of the object.

OLD The object exists.

NEW The object does not exist and window services must create it.

If *operation_type* is BEGIN and *object_type* is DSNAME, you must supply a value for *object_state*.

Define *object_state* as character data of length 3.

,access_mode

Specifies the type of access required.

READ READ access.

UPDATE

UPDATE access.

If *operation_type* is BEGIN and *object_type* is DDNAME or DSNAME, you must supply a value for *access_mode*. For a new or temporary data object, window services assumes UPDATE.

Define *access_mode* as character data of length 6. If you specify READ, pad the string on the right with 1 or 2 blanks.

,object_size

Specifies the maximum size of the new object in units of 4096 bytes.

This parameter is required if either of the following conditions is true:

- *Operation_type* is BEGIN, *object_type* is DSNAME, and *object_state* is NEW
- *Operation_type* is BEGIN and *object_type* is TEMPSPACE

Define *object_size* as integer data of length 4.

,object_id

Specifies the object identifier.

When *operation_type* is BEGIN, the service returns the object identifier in this parameter. Use the identifier to identify the object to other window services.

When *operation_type* is END, you must supply the object identifier in this parameter.

Define *object_id* as character data of length 8.

,high_offset

When CSRIDAC completes, *high_offset* contains the size of the existing object expressed in blocks of 4096 bytes

Define *high_offset* as integer data of length 4.

,return_code

When CSRIDAC completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

Return codes and reason codes are explained under “Return codes and reason codes.”

,reason_code

When CSRIDAC completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes are explained under “Return codes and reason codes.”

Abend codes

CSRIDAC issues abend code X'019'. For more information, see *z/OS MVS System Codes*.

Return codes and reason codes

When CSRIDAC returns control to your program, *return_code* contains a return code and *reason_code* contains a reason code. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. Table 2 identifies return code and reason code combinations, tells what each means, and recommends an action that you should take.

A return code of X'C' means that data-in-virtual encountered a problem or an unexpected condition. The associated reason codes are data-in-virtual reason codes. Data-in-virtual reason codes are two bytes long and right justified. To resolve a data-in-virtual problem, request help from your system programmer. For information about data-in-virtual, see the *z/OS MVS Programming: Assembler Services Guide*.

Table 2. CSRIDAC Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	Meaning: The operation was successful. Action: Continue normal program execution.
00000008 (8)	00000118 (280)	Meaning: The system could not obtain enough storage to create a hiperspace for the temporary object or the scroll area. Note: Hiperspace™ is the name the system uses to identify the storage it uses to create a temporary object or a scroll area for a permanent object. Action: Notify your system programmer. The system programmer might have to increase the SMF limit for data spaces and hiperspace that are intended for the user.
00000008 (8)	00000119 (281)	Meaning: The system could not delete or unidentify the temporary object or the scroll area. Action: Notify your system programmer.
00000008 (8)	0000011A (282)	Meaning: The system was unable to create a new VSAM linear data set. DFP 3.1 must be running and SMS must be active. Action: Notify your system programmer.
0000000C (12)	xxxx000A (10)	Meaning: Another service currently is executing with the specified ID. Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.

Table 2. CSRIDAC Return and Reason Codes (continued)

Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx001C (28)	Meaning: The object cannot be accessed at the current time. Action: Try running your program at a later time. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0037 (55)	Meaning: The caller invoked ACCESS. The access is successful, but the system is issuing a warning that the data set was not allocated with a SHAREOPTIONS(1,3). Action: Notify your system programmer.
0000000C (12)	xxxx003E (62)	Meaning: The hiperspace data-in-virtual object may not be accessed at this time. (If MODE=READ, the object is already accessed under a different ID for UPDATE. If MODE=UPDATE, the object is already accessed under at least one other ID.) Action: Try running your program at a later time. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0801 (2049)	Meaning: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. Action: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	Meaning: System error — I/O driver failure. Action: Notify your system programmer.
0000000C (12)	xxxx0805 (2053)	Meaning: System error — A system error of indeterminate origin has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	Meaning: System error — I/O from a previous request has not completed. Action: Notify your system programmer.
00000010 (16)	rrrrnnnn	Meaning: The system was unable to allocate or unallocate the data set specified as <i>object_name</i> . The value <i>rrrr</i> is the return code from dynamic allocation. The value <i>nnnn</i> is the two-byte reason code from dynamic allocation. See <i>z/OS MVS Programming: Authorized Assembler Services Guide</i> for dynamic allocation return and reason codes. Action: If <i>object_state</i> is NEW, make sure that a data set of the same name does not already exist. If one does already exist, either use the existing data set or change the name of your data set. If you are unable to correct the problem, notify your system programmer.
0000002C (44)	00000004 (4)	Meaning: Window services have not been defined to your system or the link to the service failed. Action: Notify your system programmer.

CSRREFR — Refresh an object

To refresh changed data that is in a window, a scroll area, or a temporary object, call CSRREFR. CSRREFR refreshes changed data within specified blocks as follows:

- If the object is permanent, CSRREFR replaces specified changed blocks in windows or the scroll area with corresponding blocks from the object on DASD.
- For a temporary object, CSRREFR refreshes specified changed blocks in windows and the object by setting the blocks to binary zeroes.

CSRREFR

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown. For parameters that CSRREFR uses to obtain input values, assign values that are acceptable to CSRREFR. For parameters that CSRREFR ignores, assign any value that is valid for the particular parameter's data type.

Assign values, acceptable to CSRREFR, to *object_id*, *offset*, and *span*. CSRREFR ignores other parameter values. CSRREFR returns values in *return_code* and *reason_code*.

CALL statement	Parameters
CALL CSRREFR	(object_id ,offset ,span ,return_code ,reason_code)

object_id

Specifies the object identifier. Supply the same object identifier that CSRIDAC returned when you obtained access to the object.

Define *object_id* as character data of length 8.

,offset

Specifies the offset into the object in blocks of 4096 bytes. A value of 0 specifies the first block of 4096 bytes or bytes 0 to 4095 of the object; a value of 1 specifies the second block of 4096 bytes, or bytes 4096 to 8191 of the object, and so forth.

Define *offset* as integer data of length 4.

offset and *span*, together, determine which part of the object window services refreshes. To refresh the entire object, specify 0 for *offset* and 0 for *span*.

,span

Specifies how many 4096-byte blocks CSRREFR is to refresh.

Define *span* as integer data of length 4.

,return_code

When CSRREFR completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

Return codes and reason codes are explained under "Return codes and reason codes" on page 33.

,reason_code

When CSRREFR completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes are explained under "Return codes and reason codes" on page 33.

Abend codes

CSRREFR issues abend code X'019'. For more information, see *z/OS MVS System Codes*.

Return codes and reason codes

When CSRREFR returns control to your program, *return_code* contains a return code and *reason_code* contains a reason code. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. Table 3 identifies return code and reason code combinations, tells what each means, and recommends an action that you should take.

A return code of 'XC' means that data-in-virtual encountered a problem or an unexpected condition. The associated reason codes are data-in-virtual reason codes. Data-in-virtual reason codes are two bytes long and right justified. To resolve a data-in-virtual problem, request help from your system programmer.

Table 3. CSRREFR Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	Meaning: The operation was successful. Action: Continue normal program execution.
00000008 (8)	00000152 (338)	Meaning: The system could not refresh all of the temporary object within the specified span. Action: Notify your system programmer.
0000000C (12)	xxxx000A (10)	Meaning: There is another service currently executing with the specified ID. Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	Meaning: An I/O error has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx001A (26)	Meaning: The specified range does not include any mapped block of the object. Action: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx0801 (2049)	Meaning: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. Action: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	Meaning: System error — A necessary page table could not be read into real storage. Action: Notify your system programmer.
0000000C (12)	xxxx0805 (2053)	Meaning: System error — A system error of indeterminate origin has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	Meaning: System error — I/O error. Action: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	Meaning: System error — I/O from a previous request has not completed. Action: Notify your system programmer.

Table 3. CSRREFR Return and Reason Codes (continued)

Return Code	Reason Code	Meaning and Action
0000002C (44)	00000004 (4)	<p>Meaning: Window services have not been defined to your system or the link to the service failed.</p> <p>Action: Notify your system programmer.</p>

CSRSAVE — Save changes made to a permanent object

To update specified blocks of a permanent object with changes, call CSRSAVE. The changes can be in blocks that are mapped to the scroll area, in blocks that are mapped to windows, or in a combination of these places.

Note: You cannot use CSRSAVE to save changes made to a temporary object. If you call CSRSAVE for a temporary object, CSRSAVE ignores the request and returns control to your program with a return code of 8. To save changes made to a temporary object, call CSRSCOT.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown. For parameters that CSRSAVE uses to obtain input values, assign values that are acceptable to CSRSAVE. For parameters that CSRSAVE ignores, assign any value that is valid for the particular parameter's data type.

Assign values, acceptable to CSRSAVE, to *object_id*, *offset*, and *span*. CSRSAVE ignores other parameter values. CSRSAVE returns values in *new_hi_offset*, *return_code*, and *reason_code*.

CALL statement	Parameters
CALL CSRSAVE	(<i>object_id</i> , <i>offset</i> , <i>span</i> , <i>new_hi_offset</i> , <i>return_code</i> , <i>reason_code</i>)

object_id

Specifies the object identifier. Supply the same object identifier that CSRIDAC returned when you obtained access to the object.

Define *object_id* as character data of length 8.

,offset

Specifies the offset into the object in blocks of 4096 bytes. A value of 0 specifies the first block of 4096 bytes or bytes 0 to 4095 of the object; a value of 1 specifies the second block of 4096 bytes, or bytes 4096 to 8191 of the object, and so forth.

Define *offset* as integer data of length 4.

offset and *span*, together, determine which part of the object window services saves. To save the entire object, specify 0 for *offset* and 0 for *span*.

,span

Specifies how many 4096-byte blocks CSRSAVE is to save.

Define *span* as integer data of length 4.

,new_hi_offset

When CSRSAVE completes, *new_hi_offset* contains the new size of the object expressed in units of 4096 bytes.

Define *new_hi_offset* as integer data of length 4.

,return_code

When CSRSAVE completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

Return codes and reason codes are explained under “Return codes and reason codes.”

,reason_code

When CSRSAVE completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes are explained under “Return codes and reason codes.”

Abend codes

CSRSAVE issues abend code X'019'. For more information, see *z/OS MVS System Codes*.

Return codes and reason codes

When CSRSAVE returns control to your program, *return_code* contains a return code and *reason_code* contains a reason code. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. Table 4 identifies return code and reason code combinations, tells what each means, and recommends an action that you should take.

A return code of X'4' with a reason code of X'0807' or a return code of X'C' with any reason code means that data-in-virtual encountered a problem or an unexpected condition. Data-in-virtual reason codes are two bytes long and right justified. To resolve a data-in-virtual problem, request help from your system programmer. For information about data-in-virtual, see the *z/OS MVS Programming: Assembler Services Guide*.

Table 4. CSRSAVE Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	Meaning: The operation was successful. Action: Continue normal program execution.
00000004 (4)	xxxx0807 (2055)	Meaning: Media damage may be present in allocated DASD space. The damage is beyond the currently saved portion of the object. The SAVE operation completed successfully. Action: Notify your system programmer.
00000008 (8)	xxxx0143 (323)	Meaning: You cannot use the SAVE service for a temporary object. Action: Use the scrollout (CSRSCOT) service.
0000000C (12)	xxxx000A (10)	Meaning: There is another service currently executing with the specified ID. Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.

Table 4. CSRSAVE Return and Reason Codes (continued)

Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx0017 (23)	Meaning: An I/O error has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx001A (26)	Meaning: The specified range does not encompass any mapped area of the object. Action: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx0801 (2049)	Meaning: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. Action: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	Meaning: System error — I/O driver failure. Action: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	Meaning: System error — A necessary page table could not be read into real storage. Action: Notify your system programmer.
0000000C (12)	xxxx0804 (2052)	Meaning: System error — Catalog update failed. Action: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	Meaning: System error — I/O error. Action: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	Meaning: System error — I/O from a previous request has not completed. Action: Notify your system programmer.
0000002C (44)	00000004 (4)	Meaning: Window services have not been defined to your system or the link to the service failed. Action: Notify your system programmer.

CSRSCOT — Save object changes in a scroll area

Call CSRSCOT to:

- Update specified blocks of a permanent object’s scroll area with changes that appear in a window you have defined for the object. CSRSCOT requires that the permanent object have a scroll area. CSRSCOT changes only the content of the scroll area and not the content of the permanent data object.
- Update specified blocks of a temporary data object with the changes that appear in a window you have defined for the data object.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown. For parameters that CSRSCOT uses to obtain input values, assign values that are acceptable to CSRSCOT. For parameters that CSRSCOT ignores, assign any value that is valid for the particular parameter’s data type.

Assign values, acceptable to CSRSCOT, to *object_id*, *offset*, and *span*. CSRSCOT ignores other parameter values. CSRSCOT returns values in *return_code* and *reason_code*.

CALL statement	Parameters
CALL CSRSCOT	(<i>object_id</i> , <i>offset</i> , <i>span</i> , <i>return_code</i> , <i>reason_code</i>)

object_id

Specifies the object identifier. Supply the same object identifier that CSRIDAC returned when you obtained access to the object.

Define *object_id* as character data of length 8.

,offset

Specifies the offset into the object in blocks of 4096 bytes. A value of 0 specifies the first block of 4096 bytes or bytes 0 to 4095 of the object; a value of 1 specifies the second block of 4096 bytes, or bytes 4096 to 8191 of the object, and so forth.

Define *offset* as integer data of length 4.

offset and *span*, together, determine which part of the object CSRSCOT updates. To update the entire object, specify 0 for *offset* and 0 for *span*.

,span

Specifies how many 4096-byte blocks CSRSCOT is to update.

Define *span* as integer data of length 4.

,return_code

When CSRSCOT completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

Return codes and reason codes are explained under "Return codes and reason codes."

,reason_code

When CSRSCOT completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes are explained under "Return codes and reason codes."

Abend codes

CSRSCOT issues abend code X'019'. For more information, see *z/OS MVS System Codes*.

Return codes and reason codes

When CSRSCOT returns control to your program, *return_code* contains a return code and *reason_code* contains a reason code. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. Table 5 on page 38 identifies return code and reason code combinations, tells what each means, and recommends an action that you should take.

A return code of X'C' means that data-in-virtual encountered a problem or an unexpected condition. The associated reason codes are data-in-virtual reason codes. Data-in-virtual reason codes are two bytes long and right justified. For information about data-in-virtual, see *z/OS MVS Programming: Assembler Services Guide*. To resolve the problem, request help from your system programmer.

Table 5. CSRSCOT Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	Meaning: The operation was successful. Action: Continue normal program execution.
00000004 (4)	xxxx0807 (2055)	Meaning: Media damage may be present in allocated DASD space. The damage is beyond the currently saved portion of the object. The SAVE operation completed successfully. Action: Notify your system programmer.
0000000C (12)	xxxx000A (10)	Meaning: There is another service currently executing with the specified ID. Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	Meaning: An I/O error has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx001A (26)	Meaning: The specified range does not encompass any mapped area of the object. Action: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx0801 (2049)	Meaning: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. Action: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	Meaning: System error — I/O driver failure. Action: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	Meaning: System error — A necessary page table could not be read into real storage. Action: Notify your system programmer.
0000000C (12)	xxxx0804 (2052)	Meaning: System error — Catalog update failed. Action: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	Meaning: System error — I/O error. Action: Notify your system programmer.
0000000C (12)	xxxx0808 (2056)	Meaning: System error — I/O from a previous request has not completed. Action: Notify your system programmer.
0000002C (44)	00000004 (4)	Meaning: Window services have not been defined to your system or the link to the service failed. Action: Notify your system programmer.

CSRVIEW — View an object

Call CSRVIEW to:

- Map a window to one or more blocks of a data object. If you specified scrolling when you called CSRIDAC to identify the object, CSRVIEW maps the window to the scroll area and the scroll area to the object.
- Specify that the reference pattern you are using is either random or sequential.
- End a view that you previously created through CSRVIEW or CSREVV and unmap the object.

Mapping a data object enables your program to access the data that is viewed through the window the same way it accesses other data in your storage.

The CSREVV service also maps a data object. Use that service if your program can benefit from having more than 16 blocks come into a window at one time or if it can benefit from having fewer than 16.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown. For parameters that CSRVIEW uses to obtain input values, assign values that are acceptable to CSRVIEW. For parameters that CSRVIEW ignores, assign any value that is valid for the particular parameter's data type.

The type of function you request determines which parameter values CSRVIEW uses to obtain input values:

- To map a window to a data object and begin viewing the object, specify BEGIN for *operation_type*, and assign values, acceptable to CSRVIEW, to:
 - *object_id*
 - *offset*
 - *span*
 - *window_name*
 - *usage*
 - *disposition*

CSRVIEW ignores other parameter values. CSRVIEW returns values in *return_code* and in *reason_code*.

- To end a view set by either CSRVIEW or CSREVV and to unmap the data object, specify END for *operation_type*, and assign values, acceptable to CSRVIEW, to:
 - *object_id*
 - *offset*
 - *span*
 - *window_name*
 - *usage*
 - *disposition*

CSRVIEW ignores other parameter values. CSRVIEW returns values in *return_code* and *reason_code*.

CSRVIEW

CALL statement	Parameters
CALL CSRVIEW	(operation_type ,object_id ,offset ,span ,window_name ,usage ,disposition ,return_code ,reason_code)

operation_type

Specifies the type of operation CSRVIEW is to perform. To begin viewing an object, specify BEGIN. To end a view, specify END.

Define *operation_type* as character data of length 5. If you specify END, pad the string on the right with 1 or 2 blanks.

,object_id

Specifies the object identifier. Supply the object identifier that CSRIDAC returned when you obtained access to the object.

Define *object_id* as character data of length 8.

,offset

Specifies the offset of the view into the object. Specify the offset in blocks of 4096 bytes.

Define *offset* as integer data of length 4.

,span

Specifies the window size in blocks of 4096 bytes.

Define *span* as integer data of length 4.

,window_name

Specifies the symbolic name you assigned to the window in your address space.

,usage

Specifies the expected pattern of references to pages in the object. Specify one of the following values:

SEQ The reference pattern is expected to be sequential. If you specify SEQ, window services brings up to 16 blocks of data into the window at a time, depending on the size of the window.

RANDOM

The reference pattern is expected to be random. If you specify RANDOM, window services brings data into the window one block at a time.

Define *usage* as character data of length 6. If you specify SEQ, pad the string on the right with 1 to 3 blanks.

,disposition

Defines how CSRVIEW is to handle data that is in the window when you begin or end a view.

- When you specify CSRVIEW with an *operation_type* of BEGIN and a disposition of:

REPLACE

The first time you reference a block to which the window is mapped, CSRVIEW replaces the data in the window with the data from the referenced block.

RETAIN

When you reference a block to which the window is mapped, the data in the window remains unchanged. When you call CSRSAVE to save the mapped blocks, CSRSAVE saves all of the mapped blocks because CSRSAVE considers them changed.

- When you specify CSRVIEW with an *operation_type* of END and a disposition of:

REPLACE

CSRVIEW discards the data that is in the window making the window contents unpredictable. CSRVIEW does not update mapped blocks of the object or scroll area.

RETAIN

If the object is permanent and has no scroll area, CSRVIEW retains the data that is in the window. CSRVIEW does not update mapped blocks of the object. If the object is permanent and has a scroll area, or if the object is temporary, CSRVIEW retains the data that is in the window and updates the mapped blocks of the object or scroll area.

Define *disposition* as character data of length 7. If you specify RETAIN, pad the string on the right with a blank.

,return_code

When CSRVIEW completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

Return codes and reason codes are explained under “Return codes and reason codes.”

,reason_code

When CSRVIEW completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes are explained under “Return codes and reason codes.”

Abend codes

CSRVIEW issues abend code X'019'. For more information, see *z/OS MVS System Codes*.

Return codes and reason codes

When CSRVIEW returns control to your program, *return_code* contains a return code and *reason_code* contains a reason code. Return codes and reason codes are shown in hexadecimal followed by the decimal equivalent enclosed in parentheses. Table 6 on page 42 identifies return code and reason code combinations, tells what each means, and recommends an action that you should take.

A return code of X'4' with a reason code of X'0125' or a return code of X'C' with any reason code means that data-in-virtual encountered a problem or an unexpected condition. Data-in-virtual reason codes are two bytes long and right

justified. For information about data-in-virtual, see *z/OS MVS Programming: Assembler Services Guide*. To resolve the problem, request help from your system programmer.

Table 6. CSRVIEW Return and Reason Codes

Return Code	Reason Code	Meaning and Action
00000000 (0)	00000000 (0)	Meaning: The operation was successful. Action: Continue normal program execution.
00000004 (4)	xxxx0125 (293)	Meaning: The operation was successful. The service could not retain all the data that was in the scroll area, however. Action: Notify your system programmer.
0000000C (12)	xxxx000A (10)	Meaning: There is another service currently executing with the specified ID. Action: Use a different ID or wait until the other service completes. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0017 (23)	Meaning: An I/O error has occurred. Action: Notify your system programmer.
0000000C (12)	xxxx001A (26)	Meaning: The specified range does not encompass any mapped area of the object. Action: If you expect this reason code, take whatever action the design of your program dictates. If the reason code is unexpected, check your program for errors: you might have specified the wrong range of blocks on CSRVIEW or on CSRREFR. If you do not find any errors in your program, notify your system programmer.
0000000C (12)	xxxx001C (28)	Meaning: The object cannot be accessed at the current time. Action: Try running your program at a later time. If the problem persists, notify your system programmer.
0000000C (12)	xxxx0040 (64)	Meaning: The specified MAP range would cause the hiperspace data-in-virtual object to be extended such that the installation data space limits would be exceeded. Action: Change the MAP range you have specified or request your system programmer to increase the installation's data space limits.
0000000C (12)	xxxx0801 (2049)	Meaning: System error — Insufficient storage available to build the necessary data-in-virtual control block structure. Action: Notify your system programmer.
0000000C (12)	xxxx0802 (2050)	Meaning: System error — I/O driver failure. Action: Notify your system programmer.
0000000C (12)	xxxx0803 (2051)	Meaning: System error — A necessary page table could not be read into real storage. Action: Notify your system programmer.
0000000C (12)	xxx00804 (2052)	Meaning: System error — Catalog update failed. Action: Notify your system programmer.
0000000C (12)	xxxx0806 (2054)	Meaning: System error — I/O error. Action: Notify your system programmer.

Table 6. CSRVIEW Return and Reason Codes (continued)

Return Code	Reason Code	Meaning and Action
0000000C (12)	xxxx0808 (2056)	Meaning: System error — I/O from a previous request has not completed. Action: Notify your system programmer.
0000002C (44)	00000004 (4)	Meaning: Window services have not been defined to your system or the link to the service failed. Action: Notify your system programmer.

Chapter 4. Window services coding examples

The following examples show how to invoke window services from each of the supported languages. Following each program example is an example of the JCL needed to compile, link edit, and execute the program example. Use these examples to supplement and reinforce information that is presented in other topics within this information.

Note: Included in the FORTRAN example is the code for a required assembler language program. This program ensures that the window for the FORTRAN program is aligned on a 4K boundary.

The examples are presented in Chapter 4, "Window services coding examples":

- "ADA example"
- "C/370 example" on page 50
- "COBOL example" on page 53
- "FORTRAN example" on page 57
- "Pascal example" on page 61
- "PL/I example" on page 65

ADA example

```
-- -----  
-- This program illustrates how Data Window services are invoked --  
-- using ADA. Note that the data object referenced in this program --  
-- is permanent and already allocated, and is defined by the DD --  
-- statement CSRDD1 in the JCL. --  
--  
-- This program must be linkedited with the CSR linkage-assist --  
-- routines (also known as stubs) in SYS1.CSSLIB. --  
-- -----  
  
with EBCDIC; use EBCDIC;  
with System;  
with Text_Io;  
with Unchecked_Conversion;  
with Td_Standard; use Td_Standard;  
  
procedure CRTPAN06 is  
  
    subtype Str3 is EString (1..3);  
    subtype Str5 is EString (1..5);  
    subtype Str6 is EString (1..6);  
    subtype Str7 is EString (1..7);  
    subtype Str8 is EString (1..8);  
    subtype Str9 is EString (1..9);  
  
    function Integer_Address is new Unchecked_Conversion  
        (System.Address, Integer);  
  
    function Int_To_32 is new Unchecked_Conversion  
        (Integer, Integer_32);  
  
    Orig, -- Index to indicate the 'start'  
        -- of an array  
    Ad, I : Integer; -- Temporary variables  
    Voffset, -- Offset passed as parameter  
    Vofset2, -- Offset passed as parameter
```

ADA Example

```
Vobjsiz,           -- Object size, as parameter
Vwinsiz,          -- Window size, as parameter
High_Offset,     -- Size of object in pages
New_Hi_Offset,   -- New max size of the object
Return_Code,     -- Return code
Reason_Code : Integer_32; -- Reason code
Object_Id  : Str8;  -- Identifying token
Cscroll   : Str3;  -- Scroll area YES/NO
Cobstate  : Str3;  -- Object state NEW/OLD
Coptype   : Str5;  -- Operation type BEGIN/END
Caccess   : Str6;  -- Access RANDOM/SEQ
Cusage    : Str6;  -- Usage READ/UPDATE
Cdisp     : Str7;  -- Disposition RETAIN/REPLACE
Csptype   : Str9;  -- Object type DSNAME/DDNAME/TEMPSPACE
Cobname   : Str7;  -- Object name
K         : constant Integer := 1024; -- One kilo-byte
Pagesize  : constant Integer := 4 * K; -- Page (4K) boundary
Offset    : constant Integer_32 := 0; -- Start of permanent object
Window_Size : constant Integer := 40; -- Window size in pages
Num_Win_Elem : constant Integer := Window_Size*K; -- Num of 4-byte
-- elements in window
Object_Size : constant Integer := 3*Window_Size; -- Chosen object
-- size in pages
Num_Sp_Elem : constant Integer := (Window_Size+1)*K; -- Num of
-- 4-byte elements in space

type S is array (positive range <>) of Integer; -- Define byte
-- aligned space
Sp : S (1..Num_Sp_Elem); -- Space allocated for window

procedure CSRIDAC (Op_Type      : in Str5;
                  Object_Type  : in Str9;
                  Object_Name  : in Str7;
                  Scroll_Area  : in Str3;
                  Object_State : in Str3;
                  Access_Mode  : in Str6;
                  Vobjsiz     : in Integer_32;
                  Object_Id    : out Str8;
                  High_Offset  : out Integer_32;
                  Return_Code  : out Integer_32;
                  Reason_Code  : out Integer_32);
pragma Interface (Assembler, CSRIDAC);

procedure CSRVIEW (Op_Type      : in Str5;
                  Object_Id    : in Str8;
                  Offset       : in Integer_32;
                  Window_Size  : in Integer_32;
                  Window_Name  : in S;
                  Usage        : in Str6;
                  Disposition  : in Str7;
                  Return_Code  : out Integer_32;
                  Reason_Code  : out Integer_32);
pragma Interface (Assembler, CSRVIEW);

procedure CSRSCOT (Object_Id    : in Str8;
                  Offset       : in Integer_32;
                  Span         : in Integer_32;
                  Return_Code  : out Integer_32;
                  Reason_Code  : out Integer_32);
pragma Interface (Assembler, CSRSCOT);

procedure CSRSAVE (Object_Id    : in Str8;
                  Offset       : in Integer_32;
                  Span         : in Integer_32;
                  New_Hi_Offset : out Integer_32;
                  Return_Code  : out Integer_32;
                  Reason_Code  : out Integer_32);
```

```

pragma Interface (Assembler, CSRSAVE);

procedure CSRREFR (Object_Id   : in Str8;
                  Offset      : in Integer_32;
                  Span         : in Integer_32;
                  Return_Code  : out Integer_32;
                  Reason_Code  : out Integer_32);
pragma Interface (Assembler, CSRREFR);

begin
  Text_Io.Put_Line ("<<Begin Window Services Interface Validation>>");
  Text_Io.New_Line;

  Vobjsiz := Int_To_32(Object_Size); -- Set object size in variable
  Voffset := Offset;                -- Set offset to 0 for 1st map
  Vwinsiz := Int_To_32(Window_Size); -- Set window size in variable
  Vofset2 := Offset+Vwinsiz;        -- Set offset to 40 for 2nd map

  Coptype := "BEGIN";
  Csptype := "DDNAME ";
  Cobname := "CSRDD1 ";
  Cscroll := "YES";
  Cobstate := "OLD";
  Caccess := "UPDATE";

  CSRIDAC (Coptype,           -- Set up access to the
           Csptype,          -- permanent object and
           Cobname,          -- request a scroll area
           Cscroll,
           Cobstate,
           Caccess,
           Vobjsiz,
           Object_Id,
           High_Offset,
           Return_Code,
           Reason_Code);

  -- When you want to map a window to your object, data window services
  -- expects the address of the start of the window to be on a page (4K)
  -- boundary, and the length of the window to be a multiple of 4096 bytes.
  -- If your window is an array, the address of the first element
  -- of the array must be on a page boundary. If this is not the case,
  -- you can appropriately choose one slice of your array that starts
  -- on a 4K boundary and is a multiple of 4096 bytes in length to map
  -- onto your object.
  -- To illustrate, consider the array A(1..max_len). If the address of
  -- A(1) is not on page boundary, you cannot map A(1..max_len) to your
  -- object. You can, however, map A(n..m) to your object if you choose
  -- some appropriate values n and m such that A(n) starts on a 4K
  -- boundary and A(n..m) is a multiple of 4096 bytes in length.

  Ad := Integer_Address(Sp(1)'Address); -- Get address of start of array

  -- Determine the first element whose address is on page boundary
  -- and use that element as the origin of the array.

  Orig := (Ad mod Pagesize);           -- See where the start of
                                     -- array is in page

  if Orig = 0 then                    -- If already on page boundary
    Orig := 1;                        -- Keep the old origin
  else
    Orig := (Pagesize - Orig) / 4 + 1; -- Need new origin
  end if;

  Coptype := "BEGIN";
  Cusage := "RANDOM";

```

ADA Example

```
Cdisp := "REPLACE";

-- You can pass an array slice as a parameter to a non-Ada subprogram,
-- and because the slice is a composite object, the parameter list
-- contains the actual address of the first element in the slice.
-- To elaborate further:
-- Scalar data is passed by copy, but composite data is passed by
-- reference. If the scalar value was passed as a scalar, the assembler\
-- program would receive the address of the copy and not the address of
-- the scalar. By passing the scalar value as an array slice, a
-- composite data type is being passed and thus is passed by reference.
-- Using this technique, the assembler code receives the actual address
-- of the scalar, not a copy of the scalar.

CSRVIEW (Coptype,           -- Now map a window (the array)
         Object_Id,        -- to the permanent object.
         Voffset,          -- (Actually, CSRVIEW will map the
         Vwinsiz,          -- window to the blocks in the
         Sp(Orig..Num_Sp_Elem), -- scroll area and map the scroll
         Cusage,           -- area to the object.)
         Cdisp,
         Return_Code,
         Reason_Code);

for I in 0 .. Num_Win_Elem-1 loop -- Put data in window area
    Sp(I+Orig) := I+1;
end loop;

CSRSCOT (Object_Id,        -- Capture the view in window.
         Voffset,          -- Note: only the scroll area
         Vwinsiz,          -- is updated, the permanent
         Return_Code,      -- object remains unchanged.
         Reason_Code);

Coptype := "END ";
Cusage := "RANDOM";
Cdisp := "RETAIN ";

CSRVIEW (Coptype,           -- End the view in window
         Object_Id,
         Voffset,
         Vwinsiz,
         Sp(Orig..Num_Sp_Elem),
         Cusage,
         Cdisp,
         Return_Code,
         Reason_Code);

Coptype := "BEGIN";
Cusage := "RANDOM";
Cdisp := "REPLACE";

CSRVIEW (Coptype,           -- Now map the same window
         Object_Id,        -- to different part of the
         Vofset2,          -- permanent object.
         Vwinsiz,
         Sp(Orig..Num_Sp_Elem),
         Cusage,
         Cdisp,
         Return_Code,
         Reason_Code);

for I in 0 .. Num_Win_Elem-1 loop -- Put data in window area
    Sp(I+Orig) := I+1;
end loop;
```

```

CSRSAVE (Object_Id,
         Voffset,
         Winsiz,
         New_Hi_Offset,
         Return_Code,
         Reason_Code);
-- Capture the view in window.
-- Note: this time the permanent
-- object is updated with the
-- changes.

Coptype := "END ";
CUsage := "RANDOM";
Cdisp := "RETAIN ";

CSRVIEW (Coptype,
         Object_Id,
         Voffset,
         Winsiz,
         Sp(Orig..Num_Sp_Elem),
         Cusage,
         Cdisp,
         Return_Code,
         Reason_Code);
-- End the current view in
-- the window

Coptype := "BEGIN";
Cusage := "RANDOM";
Cdisp := "REPLACE";

CSRVIEW (Coptype,
         Object_Id,
         Voffset,
         Winsiz,
         Sp(Orig..Num_Sp_Elem),
         Cusage,
         Cdisp,
         Return_Code,
         Reason_Code);
-- Now go back to reestablish
-- the 1st map using the same
-- window area

CSRREFR (Object_Id,
         Voffset,
         Winsiz,
         Return_Code,
         Reason_Code);
-- Refresh the data in the window

Coptype := "END ";
Cusage := "RANDOM";
Cdisp := "RETAIN ";

CSRVIEW (Coptype,
         Object_Id,
         Voffset,
         Winsiz,
         Sp(Orig..Num_Sp_Elem),
         Cusage,
         Cdisp,
         Return_Code,
         Reason_Code);
-- End the view in window

Coptype := "END ";
Csptype := "DDNAME ";
Cobname := "CSRDD1 ";
Cscroll := "YES";
Cobstate := "OLD";
Caccess := "UPDATE";

CSRIDAC (Coptype,
         Csptype,
         Cobname,
         Cscroll,
         Cobstate);
-- Terminate access to the
-- permanent object

```

ADA Example

```
        Caccess,
        Vwinsiz,
        Object_Id,
        High_Offset,
        Return_Code,
        Reason_Code);

end CRTPAN06;

//ADAJOB   JOB                                00000100
//* * * * *                                00000500
//* JCL USED TO COMPILE, LINK, AND EXECUTE THE ADA PROGRAM CRTPAN06 00000600
//* THAT USES DATA WINDOW SERVICES                                00000700
//* * * * *                                00000800
/*JOBPARM T=2,L=99                                             00050000
//ADACOB1 EXEC PGM=IKJEFT01,DYNAMNBR=133                      00055813
//SYSTSPT DD SYSOUT=*                                         00055913
//SYSTSIN DD *                                               00056008
        ALLOC FI(SYSLIB) DS('SYS1.CSSLIB') SHR                 00056147
        EX 'HLQ.SEVGEXE1(ADA)' 'USERID.DWS.ADA' (MAI CRE'      00056251
/*
//ADARUN   EXEC PGM=CRTPAN06,DYNAMNBR=133                      00057008
//STEPLIB DD DISP=SHR,DSN=HLQ.SEVHMOD1                        00070036
//         DD DISP=SHR,DSN=USERID.LOAD                        00100051
//         DD DISP=SHR,DSN=USERID.ADA.DWSTEST.DATA,DISP=SHR 00110051
//CSRDD1  DD DSN=USERID.ADA.DWSTEST.DATA,DISP=SHR            00120051
//CONOUT   DD SYSOUT=*,                                       00130013
//         DCB=(LRECL=133,RECFM=F)                            00140027
```

C/370 example

The following example, coded in C/370™, creates and uses a temporary data object.

```
#include <stdio.h>
#include <stdlib.h>
/* Defined macros that will be used in the program.          */
#define SIZE 8*1024
#define OBJ_SIZE 8
#define PAGE_SIZE (4*1024)
#define DWS_FILE "DWS.FILE1 "
#define TRUE 1
#define FALSE 0
char windows[SIZE];
char *view;
void init_mem(char init_value, char *low_mem, int size);
int chk_code(long int ret, long int reason, int linenumber);
main()
{
    /* Initialized variables that will be used in the Callable */
    /* Services.                                              */
    char op_type1[5] = "BEGIN";
    char op_type2[5] = "END ";
    char object_type[9] = "TEMPSPACE";
    char object_name[45] = DWS_FILE;
    char scroll_area[3] = "YES";
    char object_state[3] = "NEW";
    char access_mode[6] = "UPDATE";
    long int object_size = OBJ_SIZE;
    char disposition[7] = "REPLACE";
    char usage[6] = "SEQ ";
    char object_id[8];
    long int high_offset, return_code, reason_code;
    long int offset, window_size, window_addr;
    long int span, new_hi_offset;
    long int addr;
    int i, ret, origin, errflag = FALSE;
    double id;
    /* Set up access to a Hiperspace object using TEMPSPACE. */
}
```

```

/* Check for return code and reason code after the call. */
csridac(op_type1, object_type, object_name, scroll_area, object_state,
        access_mode,&object_size,&object_id,&high_offset,&return_code,;
        &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Define a window in a 4K region and initialize */
/* variables for CSRVIEW. Define the window for the */
/* TEMPSPACE and verify the return code and reason code. */
init_mem('0',windows,SIZE);
addr = (int) windows % 4096;
if (addr != 0) view = windows + 4096 - addr;
offset = 0; window_size = 1;
csrview(op_type1,&object_id,&offset,&window_size,view,;
        usage, disposition, &return_code, &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Change values in the window into 1. */
init_mem('1',view,4096);
/* Capture the view in the 1st window. */
offset = 0; window_size = 1;
csrscot(&object_id, &offset, &window_size,&return_code,;
        &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Make sure that CSRSAVE will not save changes for temporary */
/* object. The return code should be equal to 8 and control */
/* will be returned to the program. */
offset = 0; window_size = 1;
csrsave(&object_id, &offset, &window_size, &high_offset,;
        &return_code, &reason_code);
if (return_code != 8) {
    errflag = TRUE;
    printf("return_code was not set to proper value.\n");
}
/* Terminate the view to the window. */
offset = 0; window_size = 1;
csrview(op_type2,&object_id,&offset,&window_size,view,;
        usage, disposition, &return_code, &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Change values in the window array into 0's. */
init_mem('0',view,4096);
/* View the window again. */
offset = 0; window_size = 1;
csrview(op_type1,&object_id,&offset,&window_size,view,;
        usage, disposition, &return_code, &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* The values in the window should remain to 1's. */
for (i=0; i<4096; i++) {
    if (errflag == TRUE) printf("%d %c ", i, view[i]);
    if (view[i] != '1') errflag = TRUE;
}
/* Refresh the window to 0's. */
offset = 0; window_size = 1;
csrrefr(&object_id, &offset, &window_size,;
        &return_code, &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* The values inside the window should equal to 0's. */
for (i=0; i<4096; i++) {
    if (errflag == TRUE) printf("%d %c ", i, view[i]);
    if (view[i] != 0) errflag = TRUE;
}
/* Terminate the view to the window. */
offset = 0; window_size = 1;
csrview(op_type2,&object_id,&offset,&window_size,view,;
        usage, disposition, &return_code, &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Terminate the access to the Hiperspace object. */
csridac(op_type2, object_type, object_name, scroll_area, object_state,
        access_mode,&object_size,&object_id,&high_offset,&return_code,;

```

C/370 Example

```

        &reason_code);
chk_code(return_code,reason_code,__LINE__);
/* Report the status of the test. */
if (errflag) {
    printf("Test failed at line %d\n", __LINE__);
    exit(1);
}
else {
    printf("Test successful : %s\n", __FILE__);
    exit(0);
}
}
/* Functions that will be used in the program. */
/* chk_code will check return code and reason code returned from*/
/* the Callable Services. It will report an error if the code(s)*/
/* is not equal to 0. */
int chk_code(long int ret, long int reason, int linenumber)
{
    if (ret != 0)
        printf("return_code = %ld instead of 0 at line %d\n",
            ret, linenumber);
    if (reason != 0)
        printf("reason_code = %ld instead of 0 at line %d\n",
            reason, linenumber);
}
/* init_mem will initialize a block of memory starting at a */
/* given location to a specified value. */
void init_mem(char init_val, char *low_mem, int size)
{
    int i;
    for (i=0; i<size; i++) *(low_mem+i) = init_val;
}
/**
/**-----
/** JCL USED TO COMPILE, LINK, AND, EXECUTE THE C/370 PROGRAM
/**-----
/**
/**/DPTTST1A JOB 'DPT04P,DPT,?,S=I','DPTTST1',MSGCLASS=H,
/** CLASS=J,NOTIFY=DPTTST1,MSGLEVEL=(1,1)
/**/CC EXEC EDCC,INFILE='DPTTST1.DWS.SOURCE(DWS1)',
/** CPARAM='NOOPT,SOURCE,NOSEQ,NOMAR',
/** OUTFILE='DPTTST1.DWS.OBJECT(DWS1)'
/**-----
/** LINK STEP
/**-----
/**LKED EXEC PGM=IEWL,PARM='MAP,RMODE=ANY,AMODE=31'
/**/SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR
/** DD DSN=SYS1.CSSLIB,DISP=SHR
/**/OBJECT DD DSN=DPTTST1.DWS.OBJECT,DISP=SHR
/**/SYSLIN DD *
/** ENTRY CEESTART
/** INCLUDE OBJECT(DWS1)
/** NAME DWS1(R)
/**/SYSLMOD DD DSN=DPTTST1.DWS.LOAD,DISP=SHR
/**/SYSPRINT DD SYSOUT=*
/**/SYSUT1 DD DSN=&&SYSUT1,UNIT=SYSDA,DISP=(NEW,DELETE,DELETE),
/** SPACE=(32000,(30,30))
/**-----
/** GO STEP. THIS STEP DEFINES A NAME FOR A PERMANENT OBJECT THAT
/** THE DDNAME OBJECT TYPE WILL REFERENCE.
/**-----
/**/GO EXEC PGM=DWS1,REGION=4M
/**/STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR
/** DD DSN=DPTTST1.DWS.LOAD,DISP=SHR
/**/SYSPRINT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=125,BLKSIZE=6000)

```

```
//PLIDUMP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DD1 DD DSN=DPTTST1.DWS.FILE1,DISP=SHR
```

COBOL example

```
IDENTIFICATION DIVISION.
*****
* Program using COBOL to create a 40-page window *
* aligned on a page boundary. This is done by locating a *
* page boundary within a 40*4096+4095 byte work area. *
* The DWS interface validation routine is then called passing *
* the 40 page window. *
*****
PROGRAM-ID. DWSCBSAM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
1 WORKAREA.
2 FILLER PIC X OCCURS 167935 TIMES.
PROCEDURE DIVISION.
DISPLAY " DWSCBSAM CALLING DWSCB4K "
CALL "DWSCB4K" USING WORKAREA
DISPLAY " DWSCBSAM BACK FROM DWSCB4K "
GOBACK.
```

```
-----
IDENTIFICATION DIVISION.
PROGRAM-ID. DWSCB4K.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
1 P POINTER.
1 PR REDEFINES P PIC 9(9) COMP.
1 DUMMY PIC 9(9) COMP.
1 R PIC 9(9) COMP.
LINKAGE SECTION.
1 INWORK PIC X(167935).
1 WINDOW.
2 FILLER PIC X(4096) OCCURS 40 TIMES.
PROCEDURE DIVISION USING INWORK.
SET P TO ADDRESS OF INWORK
DIVIDE PR BY 4096
GIVING DUMMY
REMAINDER R
IF R NOT EQUAL 0 THEN
COMPUTE PR = PR + 4096 - R
SET ADDRESS OF WINDOW TO P
DISPLAY " DWSCBK4 CALLING DWSCB2 "
CALL "DWSCB2" USING WINDOW.
DISPLAY " DWSCBK4 BACK FROM DWSCB2 "
GOBACK.
```

```
-----
IDENTIFICATION DIVISION.
PROGRAM-ID. DWSCB2.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
* WINDOW SIZE CHOSEN TO BE 40 PAGES
1 NWINPG PIC 9(9) COMP VALUE 40.
1 NWINEL PIC 9(9) COMP.
1 NWLAST PIC 9(9) COMP.
1 NOBJPG PIC 9(9) COMP.
* WINDOWS WILL BEGIN ORIGIN-ING AT OFFSET 0 IN DATA OBJECT
1 WINOFF PIC 9(9) COMP VALUE 0.
1 RETRN1 PIC 9(9) COMP.
```

COBOL Example

```
1 REASON PIC 9(9) COMP.
1 NEWOFF PIC 9(9) COMP.
1 OBSIZ PIC 9(9) COMP.
1 TOKEN PIC X(8).
1 K PIC 9(9) COMP.
LINKAGE SECTION.
1 WINDOW.
  2 FILLER PIC X(4096) OCCURS 40 TIMES.
1 WINDOW-ARRAY REDEFINES WINDOW.
  2 A PIC S9(8) COMP OCCURS 40960 TIMES.
PROCEDURE DIVISION USING WINDOW.
  DISPLAY "Begin Data Windowing Services Interface Validation"
* WINDOW COMPOSED OF 4-BYTE ELEMENTS
  COMPUTE NWINEL = 1024 * NWINPG.
* WINDOW MAY NOT BEGIN AT ARRAY ELEMENT 1, SO LEAVE ROOM
  COMPUTE NWLAST = 1024 * NWINPG + 1023
* IN THE FOLLOWING, ARBITRARILY SET OBJECT SIZE = 3 WINDOWS WORTH
  COMPUTE NOBJPG = 3 * NWINPG
* SET UP ACCESS TO A HIPERSPACE OBJECT
  CALL "CSRIDAC" USING
    BY CONTENT
      "BEGIN",
      "TEMPSPACE",
      "MY FIRST HIPERSPACE",
      "YES",
      "NEW",
      "UPDATE",
    BY REFERENCE
      NOBJPG,
      TOKEN,
      OBSIZ,
      RETRN1,
      REASON
* PUT SOME DATA INTO THE WINDOW AREA
  MOVE ALL "DATA" TO WINDOW
* NOW VIEW SOMETHING IN THE WINDOW
  CALL "CSRVIEW" USING
    BY CONTENT
      "BEGIN",
    BY REFERENCE
      TOKEN,
      WINOFF,
      NWINPG,
      WINDOW,
    BY CONTENT
      "RANDOM",
      "REPLACE",
    BY REFERENCE
      RETRN1,
      REASON
* CALCULATE SOMETHING IN THE WINDOW AREA
  PERFORM VARYING K FROM 1 BY 1 UNTIL K = NWINEL
  MOVE K TO A(K)
  END-PERFORM
* CAPTURE THE VIEW IN THE WINDOW
  CALL "CSRSCOT" USING
    TOKEN,
    WINOFF,
    NWINPG,
    RETRN1,
    REASON
* END THE VIEW IN THE WINDOW
  CALL "CSRVIEW" USING
    BY CONTENT
      "END ",
    BY REFERENCE
      TOKEN,
```

```

        WINOFF,
        NWINPG,
        WINDOW,
    BY CONTENT
        "RANDOM",
        "RETAIN ",
    BY REFERENCE
        RETRN1,
        REASON
* NOW VIEW SOMETHING ELSE (2ND WINDOW'S WORTH OF DATA) IN WINDOW
    ADD NWINPG TO WINOFF
    CALL "CSRVIEW" USING
        BY CONTENT
            "BEGIN",
        BY REFERENCE
            TOKEN,
            WINOFF
            NWINPG,
            WINDOW,
    BY CONTENT
        "RANDOM",
        "RETAIN",
    BY REFERENCE
        RETRN1,
        REASON
* CALCULATE SOMETHING NEW IN THE WINDOW AREA
    PERFORM VARYING K FROM 1 BY 1 UNTIL K = NWINEL
        COMPUTE A(K) = - K
    END-PERFORM
* SAVE THE DATA IN THE WINDOW
    CALL "CSRSCOT" USING
        TOKEN,
        WINOFF,
        NWINPG,
        RETRN1,
        REASON
* NOW END THE CURRENT VIEW IN WINDOW
    CALL "CSRVIEW" USING
        BY CONTENT
            "END ",
        BY REFERENCE
            TOKEN,
            WINOFF
            NWINPG,
            WINDOW,
    BY CONTENT
        "RANDOM",
        "RETAIN ",
    BY REFERENCE
        RETRN1,
        REASON
* NOW GO BACK TO THE FIRST VIEW IN THE WINDOW
    MOVE 0 TO WINOFF
    CALL "CSRVIEW" USING
        BY CONTENT
            "BEGIN",
        BY REFERENCE
            TOKEN,
            WINOFF,
            NWINPG,
            WINDOW,
    BY CONTENT
        "RANDOM",
        "REPLACE",
    BY REFERENCE
        RETRN1,
        REASON

```

COBOL Example

```

* REFRESH THE DATA IN THE WINDOW FOR THIS VIEW
  CALL "CSRREFR" USING
    TOKEN,
    WINOFF,
    NWINPG,
    RETRN1,
    REASON
* NOW END THE VIEW IN THE WINDOW
  CALL "CSRVIEW" USING
    BY CONTENT
    "END ",
    BY REFERENCE
    TOKEN,
    WINOFF,
    NWINPG,
    WINDOW,
    BY CONTENT
    "RANDOM",
    "RETAIN ",
    BY REFERENCE
    RETRN1,
    REASON
* TERMINATE ACCESS TO THE HIPERSPACE OBJECT
  CALL "CSRIDAC" USING
    BY CONTENT
    "END ",
    "TEMPSPACE",
    "MY FIRST HIPERSPACE ENDS HERE ",
    "YES",
    "NEW",
    "UPDATE",
    BY REFERENCE
    NOBJPG,
    TOKEN,
    OBSIZ,
    RETRN1,
    REASON
  DISPLAY "-*** Run ended with Object Size in pages = " NEWOFF
  GOBACK

*****
*
*          JCL FOR COBOL EXAMPLE
*
*****
//JOB1XXX JOB 'A9907P,B9222095',
// 'A.A.USER',RD=R,
// MSGCLASS=H,NOTIFY=AAUSER,
// MSGLEVEL=(1,1),CLASS=7
//LKED EXEC PGM=IEWL,PARM='SIZE=(1024K,512K),LIST,XREF,LET,MAP',
// REGION=1024K
//SYSLIN DD DDNAME=SYSIN
//SYSLMOD DD DSN=AAUSER.USER.LOAD(CRTCON01),DISP=SHR
//SYSLIB DD DSN=CEE.SCEELED,DISP=SHR
//*
//* FF310.OBJ HOLDS OBJECT CODE FROM THE COMPILE
//*
//MYLIB DD DSN=AAUSER.FF310.OBJ,DISP=SHR
//*
//* THE CSR STUBS ARE IN SYS1.CSSLIB
//*
//INLIB DD DSN=SYS1.CSSLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  INCLUDE MYLIB(DWSCBSAM,DWSCB4K,DWSCB2)
  LIBRARY INLIB(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC)
  NAME CRTCON01(R)

```

FORTRAN example

```

*****
*
*
*   FORTRAN EXAMPLE.  THE FORTRAN EXAMPLE IS FOLLOWED BY AN
*   ASSEMBLER PROGRAM CALLED ADDR.  YOU MUST LINKEDIT THIS
*   ASSEMBLER PROGRAM WITH THE FORTRAN PROGRAM OBJECT
*   CODE AND THE CSR STUBS.  THE ASSEMBLER PROGRAM ENSURES
*   THAT YOUR WINDOW IS ALIGNED ON A 4K BOUNDARY .
*
*****
@PROCESS DC(WINCOM)
PROGRAM CRTFON01
C
C   Test Program for Data Window Services
C
C   Window size chosen to be 40 pages
PARAMETER (NWINPG = 40)
C   Window composed of 4-byte elements
PARAMETER (NWINEL = 1024*NWINPG)
C   Window may not begin at array element 1, so leave room
PARAMETER (NWLAST = 1024*NWINPG+1023)
C   In the following, arbitrarily set object size = 3 windows worth
PARAMETER (NOBJPG = 3*NWINPG)
C   Windows will begin origin-ing at offset 0 in data object
INTEGER WINOFF
PARAMETER (WINOFF = 0)
C
C   INTEGER RETRN1, REASON, HIOFF, NEWOFF, OBSIZ, OFF
INTEGER ADDR, PAGE, A
INTEGER JUNK /-1599029040/
REAL*8 TOKEN
COMMON /WINCOM/ A(NWLAST)
C
C
C   WRITE (6, 91)
91 FORMAT('1*** Begin Data Windowing Services Interface Validation')
C
C   Set up access to a Hiperspace object
CALL CSRIDAC('BEGIN',
*           'TEMPSPACE',
*           'MY FIRST HIPERSPACE',
*           'YES',
*           'NEW',
*           'UPDATE',
*           NOBJPG,
*           TOKEN,
*           OBSIZ,
*           RETRN1,
*           REASON )
C
C   Determine first page-boundary element in Window Array "A"
PAGE = ADDR(A(1))
PAGE = MOD(PAGE, 4096)
IF (PAGE .NE. 0) PAGE = (4096 - PAGE) / 4
PAGE = PAGE + 1
C
C   Put data into the window
DO 100 K = 1, NWINEL
A(K+PAGE-1) = JUNK
100 CONTINUE
C
C   Now view data in the window
CALL CSRVIEW('BEGIN',
*           TOKEN,
*           WINOFF,

```

FORTTRAN Example

```
*           NWINPG,
*           A(PAGE),
*           'RANDOM',
*           'REPLACE',
*           RETRN1,
*           REASON )
C
C   Calculate a value in the window area
DO 101 K = 1, NWINEL
  A(K+PAGE-1) = K
101 CONTINUE
C
C   Capture the view in the window
CALL CSRSCOT( TOKEN,
*           WINOFF,
*           NWINPG,
*           RETRN1,
*           REASON )
C
C   End the view in the window
CALL CSRVIEW('END ',
*           TOKEN,
*           WINOFF,
*           NWINPG,
*           A(PAGE),
*           'RANDOM',
*           'RETAIN ',
*           RETRN1,
*           REASON )
C
C   Now view other data (2nd window's worth of data) in window
CALL CSRVIEW('BEGIN',
*           TOKEN,
*           WINOFF + NWINPG,
*           NWINPG,
*           A(PAGE),
*           'RANDOM',
*           'REPLACE',
*           RETRN1,
*           REASON )
C
C   Calculate a new value in the window
DO 102 K = 1, NWINEL
  A(K+PAGE-1) = -K
102 CONTINUE
C
C   Capture the view in the window
CALL CSRSCOT( TOKEN,
*           WINOFF + NWINPG,
*           NWINPG,
*           RETRN1,
*           REASON )
C
C   Now end the current view in window
CALL CSRVIEW('END ',
*           TOKEN,
*           WINOFF + NWINPG,
*           NWINPG,
*           A(PAGE),
*           'RANDOM',
*           'RETAIN ',
*           RETRN1,
*           REASON )
C
C   Now go back to the first view in the window
CALL CSRVIEW('BEGIN',
*           TOKEN,
```

```

*          WINOFF,
*          NWINPG,
*          A(PAGE),
*          'RANDOM',
*          'REPLACE',
*          RETRN1,
*          REASON )
C
C Refresh the data in the window for this view
CALL CSRREFR( TOKEN,
*          WINOFF,
*          NWINPG,
*          RETRN1,
*          REASON )
C
C Now end the view in the window
CALL CSRVIEW('END ',
*          TOKEN,
*          WINOFF,
*          NWINPG,
*          A(PAGE),
*          'RANDOM',
*          'RETAIN ',
*          RETRN1,
*          REASON )
C
C Terminate access to the Hiperspace object
CALL CSRIDAC('END ',
*          'TEMPSPACE',
*          'MY FIRST HIPERSPACE ENDS HERE ',
*          'YES',
*          'NEW',
*          'UPDATE',
*          NOBJPG,
*          TOKEN,
*          OBSIZ,
*          RETRN1,
*          REASON )
C
STOP
END
*****
*
*
* THIS ASSEMBLER PROGRAM ENSURES THAT YOUR WINDOW IS ALIGNED *
* ON A 4K BOUNDARY. ASSEMBLE THIS PROGRAM AND LINKEDIT THE *
* OBJECT CODE WITH THE FORTRAN CODE AND THE CSR STUBS. *
*
*****
ADDR TITLE 'LOC/ADDR Function for Fortran'
*
* Calling Sequence:
*
* INTEGER ADDR
* - - -
* L = LOC(x)
* L = ADDR(x)
*
* Returns address of "x" in R0, with high-order bit set to zero
*
ADDR CSECT
ENTRY LOC
LOC EQU *
USING *,15
L 0,0(,1) Get pointer to x
N 0,MASK Set sign bit to 0
BR 14 Return

```

FORTRAN Example

```

MASK      DC      A(X'7FFFFFFF')      Mask with high-order bit 0
END
*****
*
*          JCL TO COMPILE AND LINKEDIT THE ASSEMBLER PROGRAM, THE
*          FORTRAN PROGRAM, AND THE STUBS.
*
*****
//FORTJOB JOB                                00255013
//*                                           00003100
//*                                           00003100
//* Compile and linkedit for FORTRAN        00003100
//*                                           00003100
//*                                           00003100
//VSF2CL PROC FVPGM=FORTVS2,FVREGN=2100K,FVPDECK=NODECK, 00001000
//          FVPOLST=NOLIST,FVPOPT=0,FVTERM='SYSOUT=A',    00002000
//          PGMNAME=MAIN,PGMLIB='&&GOSET',FVLNSPC=' 3200,(25,6)' 00003000
//*                                           00003100
//*          PARAMETER  DEFAULT-VALUE  USAGE              00003900
//*                                           00004000
//*          FVPGM      FORTVS2        COMPILER NAME      00005000
//*          FVREGN     2100K          FORT-STEP REGION    00006000
//*          FVPDECK   NODECK         COMPILER DECK OPTION 00007000
//*          FVPOLST   NOLIST         COMPILER LIST OPTION 00008000
//*          FVPOPT    0              COMPILER OPTIMIZATION 00009000
//*          FVTERM    SYSOUT=A       FORT.SYSTEM OPERAND 00010000
//*          FVLNSPC   3200,(25,6)    FORT.SYSLIN SPACE   00011000
//*          PGMLIB    &&GOSET        LKED.SYSLMOD DSNNAME 00012000
//*          PGMNAME   MAIN           LKED.SYSLMOD MEMBER NAME 00013000
//*                                           00014000
//FORT EXEC PGM=&FVPGM,REGION=&FVREGN,COND=(4,LT),        00015000
//          PARM='&FVPDECK,&FVPOLST,OPT(&FVPOPT)'        00016000
//STEPLIB DD DSN=HLLDS.FORT230.VSF2COMP,DISP=SHR        00017000
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=3429                 00018000
//SYSTEM DD &FVTERM                                       00019000
//SYSPUNCH DD SYSOUT=B,DCB=BLKSIZE=3440                 00020000
//SYSLIN DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,     00021000
//          SPACE=(&FVLNSPC),DCB=BLKSIZE=3200           00022000
//LKED EXEC PGM=HEWL,REGION=768K,COND=(4,LT),            00023000
//          PARM='LET,LIST,XREF'                         00024000
//SYSPRINT DD SYSOUT=A                                    00025000
//SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR                    00026000
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(200,20))             00027000
//SYSLMOD DD DSN=&PGMLIB.(&PGMNAME),DISP=(,PASS),UNIT=SYSDA, 00028000
//          SPACE=(TRK,(10,10,1),RLSE)                  00029000
//SYSLIN DD DSN=&&LOADSET,DISP=(OLD,DELETE)               00030000
//          DD DDNAME=SYSIN                              00040000
// PEND
//          EXEC VSF2CL,FVTERM='SYSOUT=H',
//          PGMNAME=CRTFON01,PGMLIB='WINDOW.USER.LOAD'    00003000
//FORT.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTFON01),DISP=SHR
//LKED.SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR                00026000
//LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,DISP=SHR,UNIT=3380,
// VOL=SER=VM2TSO
//LKED.SYSIN DD *
//          LIBRARY IN(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC,ADDR)
//          NAME CRTFON01(R)
//*
//*          The CSR stubs are available in SYS1.CSSLIB.
//*          The object code for the ADDR routine is in
//*          TEST.OBJ
//*
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR
//          DD DSN=WINDOW.TEST.OBJ,DISP=SHR
//*
//*
*****

```

```

*
*      JCL TO EXECUTE THE FORTRAN PROGRAM.
*
*****
//FON01 JOB MSGLEVEL=(1,1)
//VSF2G PROC GOPGM=MAIN,GOREGN=100K,      00001000
//      GOF5DD='DDNAME=SYSIN',          00002000
//      GOF6DD='SYSOUT=A',              00003000
//      GOF7DD='SYSOUT=B'               00004000
//*                                       00005000
//*      PARAMETER  DEFAULT-VALUE      USAGE      00007000
//*                                       00008000
//*      GOPGM     MAIN                  PROGRAM NAME 00009000
//*      GOREGN    100K                 GO-STEP REGION 00010000
//*      GOF5DD    DDNAME=SYSIN         GO.FT05F001 DD OPERAND 00011000
//*      GOF6DD    SYSOUT=A            GO.FT06F001 DD OPERAND 00012000
//*      GOF7DD    SYSOUT=B            GO.FT07F001 DD OPERAND 00013000
//*                                       00014000
//*                                       00015000
//GO      EXEC  PGM=&GOPGM,REGION=&GOREGN,COND=(4,LT) 00016000
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR      00017000
//FT05F001 DD &GOF5DD                      00018000
//FT06F001 DD &GOF6DD                      00019000
//FT07F001 DD &GOF7DD                      00020000
// PEND
//GO EXEC VSF2G,GOPGM=CRTFON01,GOREGN=999K
//GO.STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR      00017000
// DD DSN=WINDOW.USER.LOAD,DISP=SHR,VOL=SER=VM2TSO,UNIT=3380

```

Pascal example

```

*****
*
*      PASCAL example. The data object is permanent and already
*      allocated. A scroll area is used.
*
*
*
*****
program CRTPAN06;
const
  K = 1024; (* One kilo-byte *)
  PAGESIZE = 4 * K; (* 4K page boundary *)
  OFFSET = 0; (* Windows starts *)
  WINDOW_SIZE = 40; (* Window size in pages *)
  NUM_WIN_ELEM = WINDOW_SIZE*K; (* Num of 4-byte elements *)
  OBJECT_SIZE = 3*WINDOW_SIZE; (* Chosen object size in pages*)
  SPACE_SIZE = (WINDOW_SIZE+1)*4*K; (* Space allocated for window *)
type
  S = space[SPACE_SIZE] of INTEGER; (* Define byte aligned space *)
  STR3 = packed array (. 1..3 .) of CHAR;
  STR5 = packed array (. 1..5 .) of CHAR;
  STR6 = packed array (. 1..6 .) of CHAR;
  STR7 = packed array (. 1..7 .) of CHAR;
  STR9 = packed array (. 1..9 .) of CHAR;
  STR44 = packed array (. 1..44 .) of CHAR;
var
  SP : @S; (* Declare pointer to space *)
  ORIG,
  AD, I,
  VOFFSET,
  VOFSET2,
  VOBJSIZ,
  VWINSIZ,
  HIGH_OFFSET,
  NEW_HI_OFFSET,

```

Pascal Example

```

RETURN_CODE,                (* Return code          *)
REASON_CODE : INTEGER;      (* Reason code          *)
OBJECT_ID   : REAL;         (* Identifying token    *)
CSCROLL    : STR3;         (* Scroll area YES/NO  *)
COBSTATE   : STR3;         (* Object state NEW/OLD *)
COPTYPE    : STR5;         (* Operation type BEGIN/END *)
CACCESS    : STR6;         (* Access RANDOM/SEQ    *)
CUSAGE     : STR6;         (* Usage READ/UPDATE    *)
CDISP      : STR7;         (* Disposition RETAIN/REPLACE *)
CSPTYPE    : STR9;         (* Object type DSNAM/DDNAME/TEMPSPACE *)
COBNAME    : STR44;        (* Object name          *)
procedure CSRIDAC ( var OP_TYPE : STR5;
                   var OBJECT_TYPE : STR9;
                   var OBJECT_NAME : STR44;
                   var SCROLL_AREA : STR3;
                   var OBJECT_STATE : STR3;
                   var ACCESS_MODE : STR6;
                   var VOBJSIZ : INTEGER;
                   var OBJECT_ID : REAL;
                   var HIGH_OFFSET : INTEGER;
                   var RETURN_CODE : INTEGER;
                   var REASON_CODE : INTEGER); FORTRAN;
procedure CSRVIEW ( var OP_TYPE : STR5;
                   var OBJECT_ID : REAL;
                   var OFFSET : INTEGER;
                   var WINDOW_SIZE : INTEGER;
                   var WINDOW_NAME : INTEGER;
                   var USAGE : STR6;
                   var DISPOSITION : STR7;
                   var RETURN_CODE : INTEGER;
                   var REASON_CODE : INTEGER); FORTRAN;
procedure CSRSCOT ( var OBJECT_ID : REAL;
                   var OFFSET : INTEGER;
                   var SPAN : INTEGER;
                   var RETURN_CODE : INTEGER;
                   var REASON_CODE : INTEGER ); FORTRAN;
procedure CSRSAVE ( var OBJECT_ID : REAL;
                   var OFFSET : INTEGER;
                   var SPAN : INTEGER;
                   var NEW_HI_OFFSET : INTEGER;
                   var RETURN_CODE : INTEGER;
                   var REASON_CODE : INTEGER ); FORTRAN;
procedure CSRREFR ( var OBJECT_ID : REAL;
                   var OFFSET : INTEGER;
                   var SPAN : INTEGER;
                   var RETURN_CODE : INTEGER;
                   var REASON_CODE : INTEGER ); FORTRAN;
begin
  TERMOUT(OUTPUT);          (* Output to terminal   *)
  WRITELN ('<< Begin Data Windowing Services Interface Validation >>');
  WRITELN;
  VOBJSIZ := OBJECT_SIZE;  (* Set object size variable *)
  VOFFSET := OFFSET;       (* Set offset variable to 0 *)
  VWINSIZ := WINDOW_SIZE;  (* Set window size variable *)
  VOFSET2 := OFFSET+WINDOW_SIZE; (* Set offset variable to 0 *)
  COPTYPE := 'BEGIN' ;
  CSPTYPE := 'DDNAME ' ;
  COBNAME := 'CSRDD1 ' ;
  CSCROLL := 'YES' ;
  COBSTATE := 'NEW' ;
  CACCESS := 'UPDATE' ;
  CSRIDAC (COPTYPE,        (* Set up access to a    *)
           CSPTYPE,        (* hiperspace object    *)
           COBNAME,
           CSCROLL,
           COBSTATE,
           CACCESS,

```

```

        VOBJSIZ,
        OBJECT_ID,
        HIGH_OFFSET,
        RETURN_CODE,
        REASON_CODE);
NEW(SP);
AD := ADDR(SP@); (* or ORD(SP) *)
ORIG := AD mod PAGESIZE;
if ORIG <> 0 then
    ORIG := PAGESIZE-ORIG;
for I := 0 to NUM_WIN_ELEM-1 do
    SP@[4*I+ORIG] := 999999;
COPTYPE := 'BEGIN' ;
CUSAGE := 'RANDOM' ;
CDISP := 'REPLACE' ;
CSRVIEW (COPTYPE,
        OBJECT_ID,
        VOFFSET,
        VWINSIZ,
        SP@[ORIG],
        CUSAGE,
        CDISP,
        RETURN_CODE,
        REASON_CODE);
for I := 0 to NUM_WIN_ELEM-1 do
    SP@[4*I+ORIG] := I+1;
CSRSCOT( OBJECT_ID,
        VOFFSET,
        VWINSIZ,
        RETURN_CODE,
        REASON_CODE);
COPTYPE := 'END' ;
CUSAGE := 'RANDOM' ;
CDISP := 'RETAIN' ;
CSRVIEW (COPTYPE,
        OBJECT_ID,
        VOFFSET,
        VWINSIZ,
        SP@[ORIG],
        CUSAGE,
        CDISP,
        RETURN_CODE,
        REASON_CODE);
COPTYPE := 'BEGIN' ;
CUSAGE := 'RANDOM' ;
CDISP := 'REPLACE' ;
CSRVIEW (COPTYPE,
        OBJECT_ID,
        VOFSET2,
        VWINSIZ,
        SP@[ORIG],
        CUSAGE,
        CDISP,
        RETURN_CODE,
        REASON_CODE);
for I := 0 to NUM_WIN_ELEM-1 do
    SP@[4*I+ORIG] := I-101;
CSRSAVE (OBJECT_ID,
        VOFSET2,
        VWINSIZ,
        NEW_HI_OFFSET,
        RETURN_CODE,
        REASON_CODE);
COPTYPE := 'END' ;
CUSAGE := 'RANDOM' ;
CDISP := 'RETAIN' ;
CSRVIEW (COPTYPE,

```

```

(* Allocate space *)
(* Get address of space *)
(* See where space is in page *)
(* If not on page boundary *)
(* then locate page boundary *)
(* Put data into window *)
(* area *)
(* Now view data in 1st *)
(* window *)
(* Calculate a value in 1st *)
(* window *)
(* Capture the view in 1st *)
(* window *)
(* End the view in 1st window *)
(* Now view other data in the *)
(* 2nd window *)
(* Calculate a new value in *)
(* the window *)
(* End the current view in *)

```

Pascal Example

```

OBJECT_ID,                (* window          *)
VOFFSET,
VWINSIZ,
SP@[ORIG],
CUSAGE,
CDISP,
RETURN_CODE,
REASON_CODE);
COPTYPE := 'BEGIN' ;
CUSAGE := 'RANDOM' ;
CDISP := 'REPLACE' ;
CSRVIEW (COPTYPE,                (* Now go back to the view in *)
OBJECT_ID,                       (* the 1st window          *)
VOFFSET,
VWINSIZ,
SP@[ORIG],
CUSAGE,
CDISP,
RETURN_CODE,
REASON_CODE);
CSRREFR (OBJECT_ID,                (* Refresh the data in 1st  *)
VOFFSET,                          (* window                  *)
VWINSIZ,
RETURN_CODE,
REASON_CODE);
COPTYPE := 'END' ;
CUSAGE := 'RANDOM' ;
CDISP := 'RETAIN' ;
CSRVIEW (COPTYPE,                (* End the view in 1st window *)
OBJECT_ID,
VOFFSET,
VWINSIZ,
SP@[ORIG],
CUSAGE,
CDISP,
RETURN_CODE,
REASON_CODE);
COPTYPE := 'END' ;
CSPTYPE := 'DDNAME ' ;
COBNAME := 'CSRDD1 ' ;
CSCROLL := 'YES' ;
COBSTATE := 'NEW' ;
CACCESS := 'UPDATE' ;
CSRIDAC (COPTYPE,                (* Terminate access to the  *)
CSPTYPE,                          (* Hiperspace object      *)
COBNAME,
CSCROLL,
COBSTATE,
CACCESS,
VWINSIZ,
OBJECT_ID,
HIGH_OFFSET,
RETURN_CODE,
REASON_CODE);
end.
*****
*
*          JCL to compile and linkedit
*
*****
//PASC1JOB JOB                                00010005
//GO EXEC PAS22CL                             00050000
//*                                           00050102
//*      Compile and linkedit for PASCAL      00050202
//*                                           00050302
//PASC.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTPAN06),DISP=SHR 00060006
//LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,DISP=SHR,UNIT=3380, 00560000

```

```

// VOL=SER=VM2TSO                                00570000
//LKED.SYSIN DD *                                00580000
  LIBRARY IN(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC) 00590000
  NAME CRTPAN06(R)                                00600000
/*                                                00610000
/**      SYS1.CSSLIB is the source of the CSR stubs      00620002
/**                                                00650002
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR            00690000
*****
*                                                *
*      JCL to execute. A DD statement, CSRDD1, is needed to define *
*      the permanent object which already exists.          *
*                                                *
*                                                *
*****
//PASC2JOB JOB MSGLEVEL=(1,1)                    00010000
//GO EXEC PGM=CRTPAN06                            00020002
//STEPLIB DD DSN=WINDOW.PASCAL22.LINKLIB,          00030000
// DISP=SHR,UNIT=3380,                            00040000
// VOL=SER=VM2TSO                                  00050000
// DD DSN=WINDOW.USER.LOAD,                        00060000
// DISP=SHR,UNIT=3380,                            00070000
// VOL=SER=VM2TSO                                  00080000
//CSRDD1 DD DSN=DIV.TESTDS01,DISP=SHR
//OUTPUT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=133)    00090000
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=133)  00100000

```

PL/I example

```

*****
*                                                *
*      PL/I EXAMPLE                                    *
*      OBJECT IS TEMPORARY                            *
*                                                *
*                                                *
*                                                *
*****
CRTPLN3: PROCEDURE OPTIONS (MAIN);                CSR00010
DCL                                                CSR00020
(                                                CSR00030
  K INIT(1024),                                     /* ONE KILO-BYTE */ CSR00040
  PAGESIZE INIT(4096),                             /* 4K PAGE BOUNDARY */ CSR00050
  OFFSET INIT(0),                                  /* WINDOWS STARTS */ CSR00060
  WINDOW_SIZE INIT(20),                            /* WINDOW SIZE IN PAGES */ CSR00070
  NUM_WIN_ELEM INIT (20480),                        /* NUM OF 4-BYTE ELEMENTS */ CSR00080
  OBJECT_SIZE INIT (60))                            /* CHOSEN OBJECT SIZE IN PGS */ CSR00090
  FIXED BIN(31);                                    CSR00100
                                                    CSR00110
DCL                                                CSR00120
/* 32767 IS UPPER LIMIT FOR ARRAY BOUND.          */ CSR00130
S(32767) BIN(31) FIXED BASED(SP); /* DEFINE WORD ALIGNED SPACE */ CSR00140
                                                    CSR00150
DCL SP PTR;                                        CSR00160
                                                    CSR00170
DCL                                                CSR00180
(                                                CSR00190
  ORIG,                                             /* START ADDRESS OF WINDOW */ CSR00200
  AD, I,                                            /* TEMPORARY VARIABLES */ CSR00210
  HIGH_OFFSET,                                     /* SIZE OF OBJECT IN PAGES */ CSR00220
  NEW_HI_OFFSET,                                  /* NEW MAX SIZE OF THE OBJECT */ CSR00230
  RETURN_CODE,                                    /* RETURN CODE */ CSR00240
  REASON_CODE) FIXED BIN(31);                      /* REASON CODE */ CSR00250
                                                    CSR00260
DCL                                                CSR00270
OBJECT_ID CHAR(8);                                  /* IDENTIFYING TOKEN */ CSR00280
                                                    CSR00290
                                                    CSR00300

```

PL/I Example

```

/*****/ CSR00310
CSR00320
DCL CSRIDAC ENTRY(CHAR(5), /* OP_TYPE */ CSR00330
CHAR(9), /* OBJECT_TYPE */ CSR00340
CHAR(44), /* OBJECT_NAME */ CSR00350
CHAR(3), /* SCROLL_AREA */ CSR00360
CHAR(3), /* OBJECT_STATE */ CSR00370
CHAR(6), /* ACCESS_MODE */ CSR00380
FIXED BIN(31), /* OBJECT_SIZE */ CSR00390
CHAR(8), /* OBJECT_ID */ CSR00400
FIXED BIN(31), /* HIGH_OFFSET */ CSR00410
FIXED BIN(31), /* RETURN_CODE */ CSR00420
FIXED BIN(31) ) /* REASON_CODE */ CSR00430
OPTIONS(ASSEMBLER); CSR00440
CSR00450
CSR00460
DCL CSRVIEW ENTRY(CHAR(5), /* OP_TYPE */ CSR00470
CHAR(8), /* OBJECT_ID */ CSR00480
FIXED BIN(31), /* OFFSET */ CSR00490
FIXED BIN(31), /* WINDOW_SIZE */ CSR00500
FIXED BIN(31), /* WINDOW_NAME */ CSR00510
CHAR(6), /* USAGE */ CSR00520
CHAR(7), /* DISPOSITION */ CSR00530
FIXED BIN(31), /* RETURN_CODE */ CSR00540
FIXED BIN(31) ) /* REASON_CODE */ CSR00550
OPTIONS(ASSEMBLER); CSR00560
CSR00570
CSR00580
DCL CSRSCOT ENTRY(CHAR(8), /* OBJECT_ID */ CSR00590
FIXED BIN(31), /* OFFSET */ CSR00600
FIXED BIN(31), /* SPAN */ CSR00610
FIXED BIN(31), /* RETURN_CODE */ CSR00620
FIXED BIN(31) ) /* REASON_CODE */ CSR00630
OPTIONS(ASSEMBLER); CSR00640
CSR00650
CSR00660
DCL CSRSAVE ENTRY(CHAR(8), /* OBJECT_ID */ CSR00670
FIXED BIN(31), /* OFFSET */ CSR00680
FIXED BIN(31), /* SPAN */ CSR00690
FIXED BIN(31), /* NEW_HI_OFFSET */ CSR00700
FIXED BIN(31), /* RETURN_CODE */ CSR00710
FIXED BIN(31) ) /* REASON_CODE */ CSR00720
OPTIONS(ASSEMBLER); CSR00730
CSR00740
CSR00750
DCL CSRREFR ENTRY(CHAR(8), /* OBJECT_ID */ CSR00760
FIXED BIN(31), /* OFFSET */ CSR00770
FIXED BIN(31), /* SPAN */ CSR00780
FIXED BIN(31), /* RETURN_CODE */ CSR00790
FIXED BIN(31) ) /* REASON_CODE */ CSR00800
OPTIONS(ASSEMBLER); CSR00810
CSR00820
/*****/ CSR00830
CSR00840
CSR00850
PUT SKIP LIST CSR00860
(' << BEGIN DATA WINDOWING SERVICES INTERFACE VALIDATION >> '); CSR00870
PUT SKIP LIST (' '); CSR00880
CSR00890
CALL CSR00900
CSRIDAC ('BEGIN', /* SET UP ACCESS TO A HIPER- */ CSR00910
'TEMPSPACE', /* SPACE OBJECT */ CSR00920
'MY FIRST HIPERSPACE',
'YES',
'NEW',
'UPDATE',
OBJECT_SIZE,
CSR00930
CSR00940
CSR00950
CSR00960
CSR00970

```

```

OBJECT_ID,                CSR00980
HIGH_OFFSET,             CSR00990
RETURN_CODE,             CSR01000
REASON_CODE);          CSR01010
                        CSR01020
ALLOC S;                 /* ALLOCATE SPACE */ CSR01030
AD = UNSPEC(SP);         /* GET ADDRESS OF SPACE */ CSR01040
ORIG = MOD(AD,PAGESIZE); /* SEE WHERE SPACE IS IN PAGE */ CSR01050
IF ORIG ^= 0 THEN        /* IF NOT ON PAGE BOUNDARY */ CSR01060
  ORIG = (PAGESIZE-ORIG) / 4; /* THEN LOCATE PAGE BOUNDARY */ CSR01070
ORIG = ORIG + 1;        CSR01080
                        CSR01090
DO I = 1 TO NUM_WIN_ELEM; /* PUT SOME DATA INTO WINDOW */ CSR01100
  S(I+ORIG-1) = 99;      /* AREA */ CSR01110
END;                    CSR01120
                        CSR01130
CALL                    CSR01140
CSRVIEW ('BEGIN',        /* NOW VIEW DATA IN FIRST */ CSR01150
        OBJECT_ID,      /* WINDOW */ CSR01160
        OFFSET,         CSR01170
        WINDOW_SIZE,    CSR01180
        S(ORIG),        CSR01190
        'RANDOM',        CSR01200
        'REPLACE',      CSR01210
        RETURN_CODE,    CSR01220
        REASON_CODE);   CSR01230
                        CSR01240
DO I = 1 TO NUM_WIN_ELEM; /* CALCULATE VALUE IN 1ST */ CSR01250
  S(I+ORIG-1) = I+1;     /* WINDOW */ CSR01260
END;                    CSR01270
                        CSR01280
CALL                    CSR01290
CSRSCOT( OBJECT_ID,     /* CAPTURE THE VIEW IN 1ST */ CSR01300
        OFFSET,        /* WINDOW */ CSR01310
        WINDOW_SIZE,   CSR01320
        RETURN_CODE,   CSR01330
        REASON_CODE);  CSR01340
                        CSR01350
CALL                    CSR01360
CSRVIEW ('END ',        /* END THE VIEW IN 1ST WINDOW */ CSR01370
        OBJECT_ID,     CSR01380
        OFFSET,        CSR01390
        WINDOW_SIZE,   CSR01400
        S(ORIG),       CSR01410
        'RANDOM',       CSR01420
        'RETAIN ',     CSR01430
        RETURN_CODE,   CSR01440
        REASON_CODE);  CSR01450
                        CSR01460
CALL                    CSR01470
CSRVIEW ('BEGIN',      /* NOW VIEW OTHER DATA IN */ CSR01480
        OBJECT_ID,     /* 2ND WINDOW */ CSR01490
        OFFSET+WINDOW_SIZE, CSR01500
        WINDOW_SIZE,   CSR01510
        S(ORIG),       CSR01520
        'RANDOM',       CSR01530
        'REPLACE',     CSR01540
        RETURN_CODE,   CSR01550
        REASON_CODE);  CSR01560
                        CSR01570
DO I = 1 TO NUM_WIN_ELEM; /* CALCULATE NEW VALUE IN */ CSR01580
  S(I+ORIG-1) = I-101;  /* WINDOW */ CSR01590
END;                    CSR01600
                        CSR01610
CALL                    CSR01620
CSRSCOT (OBJECT_ID,    CSR01630
        OFFSET+WINDOW_SIZE, CSR01640

```

PL/I Example

```

        WINDOW_SIZE,          CSR01650
        RETURN_CODE,         CSR01670
        REASON_CODE);       CSR01680
                                CSR01690
                                CSR01700
CALL
CSRVIEW ('END ',              /* END THE CURRENT VIEW IN */ CSR01710
        OBJECT_ID,           /* WINDOW */                 CSR01720
        OFFSET+WINDOW_SIZE, CSR01730
        WINDOW_SIZE,         CSR01740
        S(ORIG),             CSR01750
        'RANDOM',             CSR01760
        'RETAIN ',          CSR01770
        RETURN_CODE,         CSR01780
        REASON_CODE);       CSR01790
                                CSR01800
                                CSR01810
CALL
CSRVIEW ('BEGIN',            /* NOW GO BACK TO THE VIEW IN */ CSR01820
        OBJECT_ID,           /* THE 1ST WINDOW */        CSR01830
        OFFSET,              CSR01840
        WINDOW_SIZE,         CSR01850
        S(ORIG),             CSR01860
        'RANDOM',             CSR01870
        'REPLACE',          CSR01880
        RETURN_CODE,         CSR01890
        REASON_CODE);       CSR01900
                                CSR01910
                                CSR01920
CALL
CSRREFR (OBJECT_ID,          /* REFRESH THE DATA IN 1ST */ CSR01930
        OFFSET,              /* WINDOW */                 CSR01940
        WINDOW_SIZE,         CSR01950
        RETURN_CODE,         CSR01960
        REASON_CODE);       CSR01970
                                CSR01980
                                CSR01990
CALL
CSRVIEW ('END ',              /* END THE VIEW IN 1ST WINDOW */ CSR02000
        OBJECT_ID,           CSR02010
        OFFSET,              CSR02020
        WINDOW_SIZE,         CSR02030
        S(ORIG),             CSR02040
        'RANDOM',             CSR02050
        'RETAIN ',          CSR02060
        RETURN_CODE,         CSR02070
        REASON_CODE);       CSR02080
                                CSR02090
                                CSR02100
CALL
CSRIDAC ('END ',              /* TERMINATE ACCESS TO THE */ CSR02110
        'TEMPSPACE',        /* HIPERSPACE OBJECT */     CSR02120
        'MY FIRST HIPERSPACE ENDS HERE ',
        'YES',               CSR02130
        'NEW',               CSR02140
        'UPDATE',           CSR02150
        WINDOW_SIZE,        CSR02160
        OBJECT_ID,          CSR02170
        HIGH_OFFSET,        CSR02180
        RETURN_CODE,        CSR02190
        REASON_CODE);       CSR02200
                                CSR02210
                                CSR02220
FREE S;                      CSR02230
END CRTPLN3;                  CSR02260
*****
*
*
*       JCL TO COMPILE AND LINKEDIT PL/I PROGRAM.
*
*
*
*****

```

```

//PLIJOB   JOB                                00010007
//*                                               00041001
//* PL/I Compile and Linkedit                 00042001
//*                                               00043001
//* Change all CRTPLNx to CRTPLNy            00044001
//*                                               00045001
//GO      EXEC PLIXCL                          00050000
//PLI.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTPLN3),DISP=SHR 00060008
//LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,UNIT=3380,VOL=SER=VM2TSO, 00070000
// DISP=SHR                                   00080000
//LKED.SYSIN DD *                              00090000
LIBRARY IN(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC) 00100001
NAME CRTPLN3(R)                               00110008
/*                                              00120000
//*                                              00121001
//*      SYS1.CSSLIB is source of CSR stubs    00130001
//*                                              00190000
//LKED.IN   DD DSN=SYS1.CSSLIB,DISP=SHR        00200000
*****
*                                               *
*                                               *
*      JCL TO EXECUTE.                         *
*                                               *
*                                               *
*                                               *
*****
//PLIRUN   JOB MSGLEVEL=(1,1)                  00010000
//*                                               00011001
//* EXECUTE A PL/I TESTCASE                   00012001
//*                                               00013001
//GO      EXEC PGM=CRTPLN3                     00020000
//STEPLIB DD DSN=WINDOW.USER.LOAD,DISP=SHR,    00030000
// UNIT=3380,VOL=SER=VM2TSO                   00040000
//SYSLIB  DD DSN=CEE.SCEERUN,DISP=SHR         00050000
//SYSABEND DD SYSOUT=*                        00070000
//SYSLOUT DD SYSOUT=*                         00080000
//SYSPRINT DD SYSOUT=*                       00090000

```

PL/I Example

Part 2. Reference pattern services

Chapter 5. Introduction to reference pattern services

Reference pattern services allow HLL programs to define a reference pattern for a specified area of virtual storage that the program is about to reference. Additionally, the program specifies how much data it wants the operating system to bring into central storage at one time. Data and instructions in virtual storage must reside in central storage before they can be processed. The system honors the request according to the availability of central storage. By bringing in more data at one time, the system might improve the performance of your program.

The term reference pattern refers to the order in which a program's instructions process a range of data, such as an array or part of an array.

Programs that benefit most from reference pattern services are those that reference amounts of data that are greater than one megabyte. The program should reference the data in a sequential manner and in a consistent direction, either forward or backward. In forward direction, the program references data elements in order of ascending addresses. In backward direction, the program references data elements in order of decreasing addresses. In addition, if the program "skips over" certain areas, and these areas are of uniform size and are repeated at regular intervals throughout the area, reference pattern services might provide additional performance improvement.

Two reference pattern services are available through program CALLs:

- CSRIRP identifies the range of data and the reference pattern, and defines the number of bytes that the system is requested to bring into central storage at one time. These activities are called "defining the reference pattern".
- CSRRRP removes the definition; it tells the system that the program has stopped using the reference pattern with the range of data.

A program might have a number of different ways of referencing a particular area. In this case, the program can issue multiple pairs of CSRIRP and CSRRRP services for the area. Only one pattern can be in effect at a time.

Although reference pattern services can be used for data structures other than arrays, for simplicity, examples in Chapter 5, "Introduction to reference pattern services" and Chapter 6, "Using reference pattern services," on page 79 use the services with arrays.

How does the system manage data?

Before you can evaluate the performance advantage that reference pattern services offer, you must understand some facts about how the operating system handles the data your program references. The system divides the data into 4096-byte chunks; each chunk is called a "page". For the processor to execute an instruction, the page that contains the data that the instruction requires must reside in central storage. Central storage contains pages of data for many programs — your program, plus other programs that the system is working on. The system brings a page of your data into central storage when your program needs data on that page. If the program uses the data in a sequential manner, once the program finishes using the data on that page, it will not immediately use the page again. After your program finishes using that page, the system might remove the page from central storage to

make room for another page of your data or maybe a page of some other program's data. The system allows pages to stay in central storage if they are referenced frequently enough and if the system does not need those pages for other programs.

The process that the system goes through when it pauses to bring a page into central storage is called a "page fault". This interruption causes the system to stop working on your program (or "suspend" your program) while more of your program's data comes into central storage. Then, when the page is in central storage and the system is available to your program again, the system resumes running your program at the instruction where it left off.

Reference pattern services can change the way the system handles your program's data. With direction from reference pattern services, the system moves multiple pages into central storage at a time. By bringing in many pages at a time, the system takes fewer page faults. Fewer page faults mean possible performance gains for your program.

An example of how the system manages data in an array

To evaluate the performance advantage reference pattern services offers, you need to understand how the system handles a range of data. The best way to describe this is through an example of a simple two-dimensional array. As array A(i,j) of 3 rows and 4 columns illustrates, the system stores arrays in FORTRAN programs in column-major order and stores arrays in COBOL, Pascal, PL/1, and C programs in row-major order.

- A(1,1) A(1,2) A(1,3) A(1,4)
- A(2,1) A(2,2) A(2,3) A(2,4)
- A(3,1) A(3,2) A(3,3) A(3,4)

The system stores the elements of the arrays in the following order:

Sequence of Element in Storage	FORTRAN Array Element	COBOL, Pascal, PL/1, C Array Element
1	A(1,1)	A(1,1)
2	A(2,1)	A(1,2)
3	A(3,1)	A(1,3)
4	A(1,2)	A(1,4)
5	A(2,2)	A(2,1)
6	A(3,2)	A(2,2)
7	A(1,3)	A(2,3)
8	A(2,3)	A(2,4)
9	A(3,3)	A(3,1)
10	A(1,4)	A(3,2)
11	A(2,4)	A(3,3)
12	A(3,4)	A(3,4)

Examples in Chapter 5, "Introduction to reference pattern services," on page 73 and Chapter 6, "Using reference pattern services," on page 79 depict data as a horizontal string. The elements in the arrays, therefore, would look like the following:

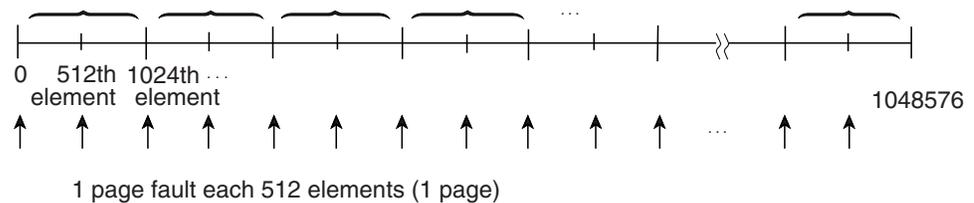
Location of elements											
1	2	3	4	5	6	7	8	9	10	11	12

Consider a two-dimensional array, ARRAY1, that has 1024 columns and 1024 rows and each element is eight bytes in size. The size of the array, therefore, is 1048576 elements or 8388608 bytes. For simplicity, assume the array is aligned on a page

boundary. Also, assume the data is not in central storage. The program references each element in the array in a forward direction, starting with the first element.

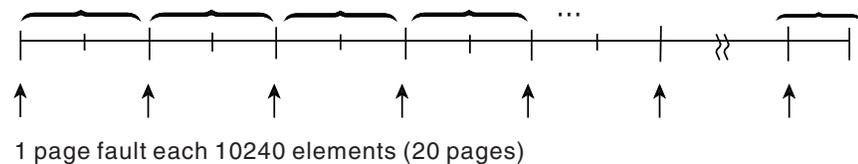
First, consider how the system brings data into central storage without information from reference pattern services. At the first reference of ARRAY1, the system takes a page fault and brings into central storage the page (of 4096 bytes) that contains the first element. After the program finishes processing the 512th (4096 divided by 8) element in the array, the system takes another page fault and brings in a second page. The system takes a page fault every 512 elements, throughout the array.

The following linear representation shows the elements in the array and the page faults the system takes as a program processes the array.



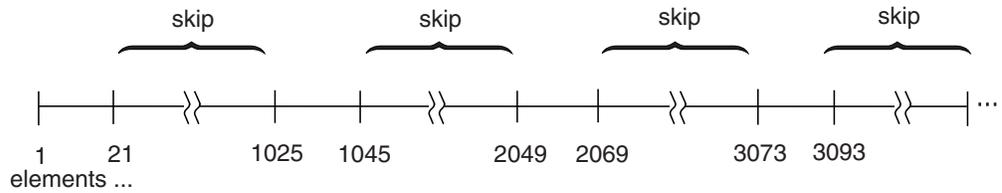
By bringing in one page at a time, the system takes 2048 page faults (8388608 divided by 4096), each page fault adding to the elapsed time of the program.

Suppose, through CSRIRP, the system knew in advance that a program would be using the array in a consistently forward direction. The system could then assume that the program's use of the pages of the array would be sequential. To decrease the number of page faults, each time the program requested data that was not in central storage, the system could bring in more than one page at a time. Suppose the system brought the next 20 consecutive pages (81920 bytes) of the array into central storage on each page fault. In this case, the system takes not 2048 page faults, but 103 (8388608 divided by 81920=102.4). Page faults occur in the array as follows:



The system brings in successive pages only to the end of the array.

Consider another way of referencing ARRAY1. The program references the first twenty elements, then skips over the next 1004 elements, and so forth through the array. CSRIRP allows you to tell the system to bring in only the pages that contain the data the program references. In this case, the reference pattern includes a repeating gap of 8032 bytes (1004×8) every 8192 bytes (1024×8). The pattern looks like this:



The grouping of consecutive bytes that the program references is called a **reference unit**. The grouping of consecutive bytes that the program skips over is called a **gap**. Reference units and gaps alternate throughout the array at regular intervals. The reference pattern is as follows:

- The reference unit is 20 elements in size — 160 consecutive bytes that the program references.
- The gap is 1004 elements in size — 8032 consecutive bytes that the program skips over.

Figure 7 shows this reference pattern and the pages that the system does not bring into central storage.

What pages does the system bring in when a gap exists?

When a gap exists, the number of pages the system brings in depends on the size of the gap, the size of the reference unit, and where the page boundary lies in relation to the gap and the reference unit. The following examples illustrate those factors.

Example 1

Figure 7 illustrates ARRAY1, the 1024-by-1024 array of eight-byte elements, where the program references 20 elements, then skips over the next 1004, and so forth in a forward direction throughout the array. The reference pattern includes a reference unit of 160 and a gap of 8032 bytes. The reference units begin on every other page boundary.

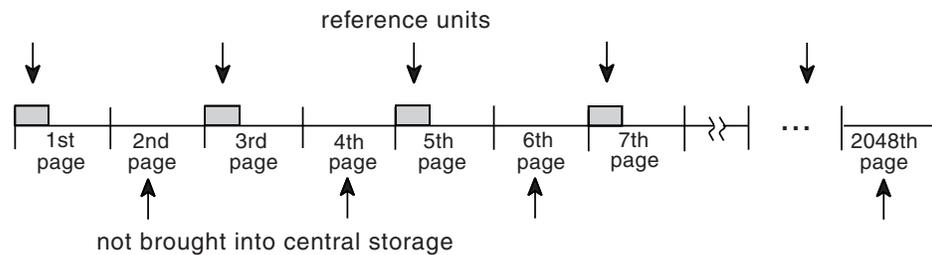
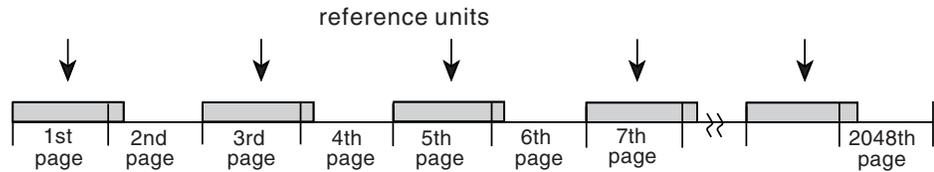


Figure 7. Illustration of a Reference Pattern with a Gap

Every other consecutive page of the data does not come into central storage; those pages contain only the “skipped over” data.

Example 2

In example 2, the reference pattern includes a reference unit of 4800 bytes and a gap of 3392 bytes. The example assumes that the area to be referenced starts on a page boundary.

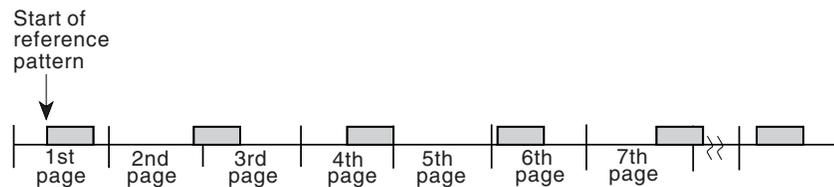


all pages brought into central storage

Because each page contains data that the program references, the system brings in all pages.

Example 3

In example 3, the area to be referenced does not begin on a page boundary. The reference pattern includes a reference unit of 2000 bytes and a gap of 5000 bytes. When you specify a reference pattern that includes a gap, the reference unit must be at the start of the area, as the following illustration shows:



most pages brought into central storage

Because the gap is larger than 4096 bytes, some pages do not come into central storage. Notice that the system does not bring in the fifth page.

Summary of how the size of the gap affects the number of pages the system brings into central storage:

- If the gap is less than 4096 bytes, the system has to bring into central all pages of the array.
- If the gap is greater than 4095 bytes and less than 8192, the system might not have to bring in certain pages. Pages that contain only data in the gap do not come in.
- If the gap is greater than 8191 bytes, the system definitely does not have to bring in certain pages that contain the gap.

Chapter 6. Using reference pattern services

The two reference pattern services are CSRIRP and CSRRRP. First, you issue CALL CSRIRP to define a reference pattern for an area; then, issue CALL CSRRRP to remove the definition of reference pattern for the area. To avoid unnecessary processing, issue the calls outside of the loops that control processing of the data elements contained in the area.

Defining the reference pattern for a data area

On CSRIRP, you tell the system:

- The lowest address of the area to be referenced
- The size of the area
- The direction of reference
- The reference pattern, in terms of reference unit and gap (if one exists)
- The number of reference units the system is to bring into central storage on a page fault

The system will not process CSRIRP unless the values you specify can result in a performance gain for your program. To make sure the system processes CSRIRP, ask the system to bring in more than three pages (that is, 12288 bytes) on each page fault.

Your program can have only one pattern defined for that area at one time. If your program will later reference the same area with another reference pattern, use CSRRRP to remove the definition, and then use CSRIRP to define another pattern.

Although the system brings in pages 4096 bytes at a time, you do not have to specify values on CSRIRP or CSRRRP in increments of 4096.

Defining the range of the area

On CSRIRP, you define the range of the area to be referenced:

- *low_address* identifies the lowest addressed byte in the range.
- *size* identifies the size, in bytes, of the range.

When reference is forward, *low_address* identifies the first element that the program can reference in the range. When reference is backward, *low_address* identifies the last element that the program can reference in the range: reference proceeds from the high-address end in the range towards *low_address*.

The following parameters define the lowest address and the size of ARRAY1, a 1024-by-1024 array that consists of 8-byte elements. ARRAY1(1,1) identifies the element in the first row and the first column.

CSRIRP with low_address of ARRAY1(1,1)
size of 1024*1024*8 bytes

When a gap exists, define the range according to the following rules:

- If direction is forward, *low_address* must be the first data element in a reference unit.

- If direction is backward, the value you use for *size* must be such that the first data element the program references is the high-address end of a reference unit.

These two rules are described and illustrated in “Using CSRIRP when a gap exists” on page 81.

Identifying the direction of the reference

On *direction*, you specify the direction of reference through the array. Forward reference means instructions start with the element indicated by *low_address* and proceed through the range of data specified by *size*. Backward reference means the program starts processing the high-address end of the range specified by *size* and proceeds toward the *low_address* end.

- “+1” indicates forward direction.
- “-1” indicates backward direction.

An example of forward reference through ARRAY1 is specified as follows:

CSRIRP with direction of +1

“Using CSRIRP when a gap exists” on page 81 contains examples of forward and backward references when a gap exists.

Defining the reference pattern

Figure 8 identifies two reference patterns that characterize most of the reference patterns that reference pattern services applies to.

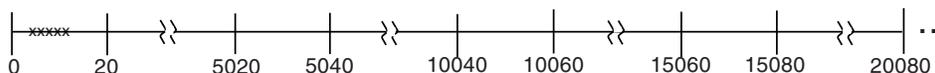
Pattern #1: No uniform gap



Characteristics of pattern:

- No uniform gap
- Reference in regular intervals (such as every element) or in irregular intervals

Pattern #2: Uniform gap



Characteristics of pattern:

- Gaps of uniform size
- Reference units, uniform in size, that occur in a repeating pattern

Figure 8. Two Typical Reference Patterns

How you define the reference pattern depends on whether your program’s reference pattern is like pattern #1 or pattern #2.

- With pattern #1 where no uniform gap exists, the program uses every element, every other element, or at least most elements on each page of array data. No definable gap exists. Do not use reference pattern services if the reference pattern is irregular and includes skipping over many areas larger than a page.
 - The *unitsize* parameter identifies the reference pattern; it indicates the number of bytes you want the system to use as a reference unit. Look at logical groupings of bytes, such as one row, a number of rows, or one element, if the

elements are large in size. Or, you might choose to divide the area to be referenced, and bring in that area on a certain number of page faults. Use the value 0 on *gapsize*.

- The *units* parameter tells the system how many reference units to try to bring in on a page fault. For a reference pattern that begins on a page boundary and has no gaps, the total number of bytes the system tries to bring into central storage at a time is the value on *unitsize* times the number on *units*, rounded up to the nearest multiple of 4096. See “Choosing the number of bytes on a page fault” on page 82 for more information on how to choose the total number of bytes.
- With pattern #2 where a uniform gap exists, the pattern includes alternating gaps and reference units. Specify the reference pattern carefully. If you identify a reference pattern and do not adhere to it, the system will work harder than if you had not used the service.
 - The *unitsize* and *gapsize* parameters identify the reference pattern. Pattern #2 in Figure 8 on page 80 includes a reference unit of 20 bytes and a gap of 5000 bytes. Because the gap is greater than 4095, some pages of the array might not be brought into central storage.
 - The *units* parameter tells the system how many reference units to try to bring into central storage at a time. “What pages does the system bring in when a gap exists?” on page 76 can help you understand how many bytes come into central storage at one time when a gap exists.

Using CSRIRP when a gap exists

When a gap exists, you have to follow one of two rules in coding the two parameters, *low_address* and *size*, that define the range of data. The direction of reference determines which rule you follow:

- When reference is forward, *low_address* must identify the beginning of a reference unit.

Figure 9 illustrates forward reference through a range of data that includes gaps. Consider the reference pattern where the program references 2000 bytes and skips the next 5000 bytes, and so forth throughout the array. The range of data starts at *low_address* and ends at the point identified in the figure by A. A can be any part of a gap or reference unit.

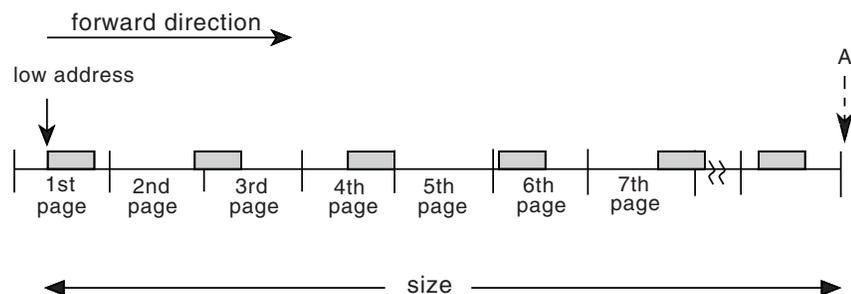


Figure 9. Illustration of Forward Direction of Reference

- When reference is backward, the value you code on *size* determines the location of the first element the program actually references. Calculate that value so that the first element the program references is the high-address end of a reference unit.

Figure 10 on page 82 illustrates backward reference through the same array as in Figure 9. Again, the program references 2000 bytes and skips the next 5000 bytes, and so forth throughout the array. The range starts at *low_address* and ends at

the point identified in the figure by **B**, where **B** must be the high-address end of a reference unit. *low_address* can be any part of a gap or reference unit.

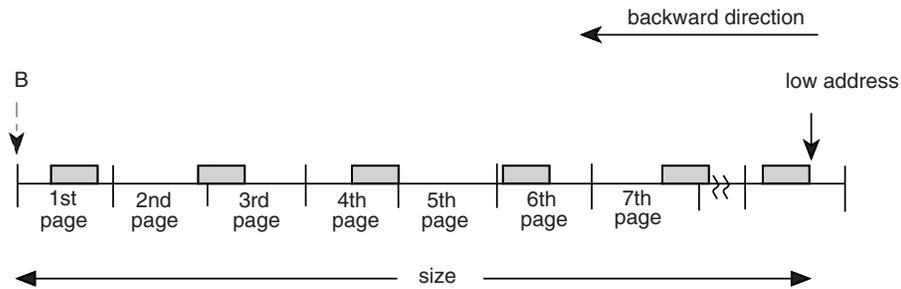


Figure 10. Illustration of Backward Direction of Reference

Choosing the number of bytes on a page fault

An important consideration in using reference pattern services is how many bytes to ask the system to bring in on a page fault. To determine this, you need to understand some factors that affect the performance of your program.

Pages do not stay in central storage if they are not referenced frequently enough and other programs need that central storage. The longer it takes for a program to begin referencing a page in central storage, the greater the chance that the page has been moved out before being referenced. When you tell the system how many bytes it should try and bring into central at one time, you have to consider the following:

1. Contention for central storage:

Your program contends for central storage along with all other submitted jobs. The greater the size of central storage, the more bytes you can ask the system to bring in on a page fault. The system responds with as much of the data you request as possible, given the availability of central storage.

2. Contention for processor time:

Your program contends for the processor's attention along with all other submitted jobs. The more competition, the less the processor can do for your program and the smaller the number of bytes you should request.

3. The elapsed time of processing one page of your data:

How long it takes a program to process a page depends on the number of references per page and the elapsed time per reference. If your program uses only a small percentage of elements on a page and references them only once or twice, the program completes the use of pages quickly. If the processing of each referenced element includes processor-intensive operations or a time-intensive operation, such as I/O, the time the program takes to process a page increases.

Conditions might vary between the peak activity of the daytime period and the low activity of the nighttime. You might be able to request a greater number at night than during the day.

What if you specify too many bytes? What if you ask the system to bring in so many pages that, by the time your program needs to use some of those pages, they have left central storage? The answer is that the system will have to bring them in again. This action causes an extra page fault and extra system overhead and decreases the benefit of reference pattern services.

For example, suppose you ask the system to bring in 204800 bytes, or 50 pages, at a time. But, by the time your program begins referencing the data on the 30th page, the system has moved that page and the ones after it out of central storage. It moved them out because the program did not use them soon enough. In this case, your program has lost the benefit of moving the last 21 pages in. Your program would get more benefit by requesting fewer than 30 pages.

What if you specify too few bytes? If you specify too small a number, the system will take more page faults than it needs to and you are not taking full advantage of reference pattern services.

For example, suppose you ask the system to bring in 40960 bytes (or 10 pages) at a time. Your program's use of each page is not time-intensive, meaning that the program finishes using the pages quickly. The program can request a number greater than 10 without causing additional page faults.

IBM recommends that you use one of the following approaches, depending on whether you want to involve your system programmer in the decision.

- The first approach is the simple one. Choose a conservative number of bytes, around 81920 (20 pages), and run the program. Look for an improvement in the elapsed time. If you like the results, you might increase the number of bytes. If you continue to increase the number, at some point you will notice a diminishing improvement or even an increase in elapsed time. Do not ask for so much that your program or other programs suffer from degraded performance.
- The second approach is for the program that needs very significant performance improvements — those programs that require amounts in excess of 50 pages. If you have such a program, you and your system programmer should examine the program's elapsed time, paging speeds, and processor execution times. In fact, the system programmer can tune the system with your program in mind, providing the needed paging resources. *z/OS MVS Initialization and Tuning Guide* can provide information on tuning the system.

Reference pattern services affects movement of pages from auxiliary **and** expanded storage to central storage. To gain insight into the effectiveness of your reference patterns, you and your system programmer will need the kind of information that the SMF Type 30 record provides. A Type 30 record includes counts of pages moved in anticipation of your program's use of those pages. The record provides counts of pages moved between expanded and central and between auxiliary and central. It also provides elapsed time values. Use this information to calculate rates of movement in determining whether to specify a very large number of bytes — for example, amounts greater than 204800 bytes (50 pages).

Examples of using CSRIRP to define a reference pattern

To clarify the relationships between the *unitsize*, *gapsize*, and *units* parameters, this topic contains three examples of defining a reference pattern. So that you can compare the three examples with what the system does without information from CSRIRP, the following call approximates the system's normal paging operation:

```
CSRIRP with unitsize of 4096 bytes  
           gapsize of 0 bytes  
           units of 1 reference unit (that is, one page)
```

Each time the system takes a page fault, it brings in 4096 bytes (one page), the system's reference unit. It brings in one reference unit at a time.

Example 1 The program processes all elements in an array in a forward direction. The processing of each element is fairly simple. The program runs during the peak hours, and many programs compete for processor time and central storage. A reasonable value to choose for the number of bytes to come into central on a page fault might be 80000 bytes (around 20 pages); *unitsize* can be 4000 bytes and *units* can be 20. The following CSRIRP service communicates this pattern to the system:

```
CSRIRP with unitsize of 4000 bytes
           gapsize of 0 bytes
           units of 20
           direction of +1
```

Example 2 The program performs the same process as in Example 1, except the program does not reference every element in the array. The program runs during the night hours when contention for the processor and for central storage is light. In this case, a reasonable value to choose for the number of bytes to come into central storage on a page fault might be 200000 bytes (around 50 pages). *unitsize* can again be 4000 bytes and *units* can be 50. The following CSRIRP service communicates this pattern:

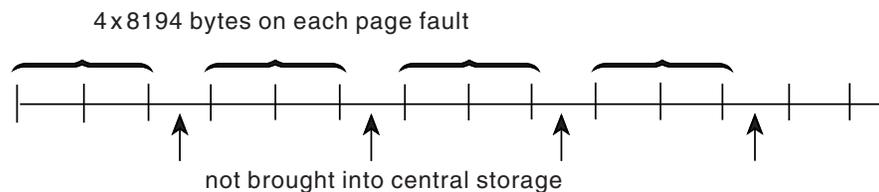
```
CSRIRP with unitsize of 4000 bytes
           gapsize of 0 bytes
           units of 50
           direction of +1
```

Example 3 The program references in a consistently forward direction through the same large array. The pattern of reference in this example includes a gap. The program references 8192 bytes, then skips the next 4096 bytes, references the next 8192 bytes, skips the next 4096 bytes throughout the array. The program chooses to bring in data 8 pages at a time. Because of the placement of reference units and gaps on page boundaries, the system does not bring in the data in the gaps.

The following CSRIRP service reflects this reference pattern:

```
CSRIRP with unitsize of 4096*2 bytes
           gapsize of 4096 bytes
           units of 4
           direction of +1
```

where the system is to bring into central storage 8 pages (4×4096×2 bytes) on a page fault. The system's response to CSRIRP is illustrated as follows:



Removing the definition of the reference pattern

When a program is finished referencing the array in the way you specified on CSRIRP, use CSRRRP to remove the definition. The following example tells the system that the program in "Defining the range of the area" on page 79 has stopped referencing the array. *low_address* and *size* have the same values you coded on the CSRIRP service that defined the reference pattern for that area.

```
CSRRRP with low_address of ARRAY1(1,1)
           size of 1024*1024*8 bytes
```

Handling return codes

Each time you call CSRIRP or CSRRRP, your program receives a return code and a reason code. These codes indicate whether the service completed successfully or whether the system rejected the service.

When you receive a return code that indicates a problem or an unusual condition, try to correct the problem, and rerun the program. Return codes and reason codes are described in Chapter 7, “Reference pattern services,” on page 87 with the description of each reference pattern service.

Chapter 7. Reference pattern services

To use reference pattern services, you issue CALLs that invoke the appropriate reference pattern services program. Each service program performs one or more functions and requires a set of parameters coded in a specific order on the CALL statement.

This topic describes the CALL statements that invoke reference pattern services. Each description includes a syntax diagram, parameter descriptions, and return code and reason code explanations with recommended actions. For examples of how to code the CALL statements, see Chapter 8, “Reference pattern services coding examples,” on page 91.

This topic contains the following subtopics:

- “CSRIRP — Define a reference pattern”
- “CSRRRP — Remove a reference pattern” on page 89.

CSRIRP — Define a reference pattern

Call CSRIRP to define a reference pattern for a large data area, such as an array, that you are about to reference. Through CSRIRP, you identify the data area and describe the reference pattern. Additionally, you tell the system how many bytes of data you want it to bring into central storage on a page fault (that is, each time the program references data that is not in central storage). This action might significantly improve the performance of the program.

Two parameters define the reference pattern:

- *unitsize* refers to a reference unit — a grouping of consecutive bytes that the program references.
- *gapsize* refers to a gap — a grouping of consecutive bytes that the program repeatedly skips over; when a pattern has a gap, reference units and gaps alternate throughout the data area.

Reference units and gaps must each be uniform in size and appear throughout the data area at repeating intervals.

Another parameter, *units*, allows you to specify how many reference units you want the system to bring into central storage each time the program references data that is not in central storage.

When you end the reference pattern in that data area, call the CSRRRP service.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown. For parameters that CSRIRP uses to obtain input values, assign appropriate values.

On entry to CSRIRP, register 1 points to the reference pattern service parameter list. Note that when a FORTRAN program calls CSRIRP, and it is running in access register (AR) mode, register 1 does not point to the reference pattern service parameter list; it points to a list of parameter addresses. Each address in this list points to the data in the corresponding parameter of the reference pattern service parameter list. To use reference pattern services in this environment, the caller

must provide an assembler interface routine to convert the FORTRAN parameter list to the form expected by reference services.

Assign values, acceptable to CSRIRP, to *low_address*, *size*, *direction*, *unitsize*, *gapsize*, and *units*. CSRIRP returns values in *return_code* and *reason_code*.

CALL statement	Parameters
CALL CSRIRP	(low_address ,size ,direction ,unitsize ,gapsize ,units ,return_code ,reason_code)

The parameters are explained as follows:

low_address

Specifies the beginning point of the data to be referenced.

low_address is the name of the data that resides at the beginning of the data area. When the direction is forward and a gap exists, *low_address* must identify the beginning of a reference unit.

,size

Identifies the *size*, in bytes, of the data area to be accessed. When direction is backward and a gap exists, the value of *size* must be such that the first data element the program references is the high-address end of a reference unit.

Define *size* as integer data of length 4.

,direction

Indicates the direction of reference, either "+1" for forward or "-1" for backward.

Define *direction* as integer data of length 4.

,unitsize

Specifies the size of a reference unit.

If the pattern does not have a gap, define the reference unit as a logical grouping according to the structure of the data array. Examples are: one row, a number of rows, one element, or one page (4096 bytes). If the pattern has a gap, define *unitsize* as the grouping of bytes that the program references and *gap* as the grouping of bytes that the program skips over.

Define *unitsize* as integer data of length 4.

,gapsize

Specifies the size, in bytes, of a gap. If the pattern has a gap, define the gap as the grouping of bytes that the program skips over. If the pattern does not have a gap, use the value "0".

Define *gapsize* as integer data of length 4.

,units

Indicates how many reference units the system is to bring into central storage each time the program needs data that is not in central storage.

Define *units* as integer data of length 4.

,return_code

When CSRIRP completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

,reason_code

When CSRIRP completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes

When CSRIRP returns control to your program, *return_code* contains a return and *reason_code* contains a reason code. The following table identifies return code and reason code combinations and tells what each means.

Return and reason codes, in hexadecimal, from CSRIRP are:

Return Code	Reason Code	Meaning
00	None	CSRIRP completed successfully.
04	xx0001xx	CSRIRP completed successfully; however, the system did not accept the reference pattern the caller specified. The system decided that bringing in pages of 4096 bytes would be more efficient.
08	xx0002xx	Unsuccessful completion. The range that the caller specified overlaps the range that a previous request specified.
08	xx0003xx	Unsuccessful completion. The number of CSRIRP requests for the user exceeds 100, the maximum number the system allows.
08	xx0004xx	Unsuccessful completion. Storage is not available for the CSRIRP service.
08	00000004	Unsuccessful completion. The direction that the caller specified is not valid.

CSRRRP — Remove a reference pattern

Call CSRRRP to remove the reference pattern for a data area, as specified by the CSRIRP service. On CSRRRP, you identify the beginning of the data area and its size. Code *low_address* and *size* exactly as you coded them on the CSRIRP service that defined the reference pattern.

Code the CALL following the syntax of the high-level language you are using and specifying all parameters in the order shown. For parameters that CSRRRP uses to obtain input values, assign values that are acceptable to CSRRRP.

Assign values to CSRRRP, to *low_address* and *size*. CSRRRP returns values in *return_code* and *reason_code*.

CSRRRP

CALL statement	Parameters
CALL CSRRRP	(<i>low_address</i> , <i>size</i> , <i>return_code</i> , <i>reason_code</i>)

The parameters are explained as follows:

low_address

Specifies the beginning point of the data to be referenced.

low_address is the name of the data that resides at the beginning of the data area.

,size

Specifies the size, in bytes, of the data area.

Define *size* as integer data of length 4.

,return_code

When CSRRRP completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

,reason_code

When CSRRRP completes, *reason_code* contains the reason code. Define *reason_code* as integer data of length 4.

Return codes and reason codes

When CSRRRP returns control to your program, *return_code* contains a hexadecimal return code and *reason_code* contains a hexadecimal reason code. The following table identifies return code and reason code combinations and tells what each means.

Return Code	Reason Code	Meaning
00	None	CSRRRP completed successfully.
08	xx0101xx	Unsuccessful completion. No CSRIRP service request was in effect for the specified data area. Check to see if the system rejected the previous CSRIRP request for the data area.

Chapter 8. Reference pattern services coding examples

The following examples show how to invoke reference pattern services from each of the supported languages. Following each program example is an example of the JCL needed to compile, link edit, and execute the program example. Use these examples to supplement and reinforce information that is presented in other topics within this information.

Note: Included in the FORTRAN example is the code for a required assembler language program. This program ensures that the reference pattern for the FORTRAN program is aligned on a 4K boundary.

The programs in this topic are similar. They each process two arrays, A and B. The arrays are 200×200 in size, each element consisting of 4 bytes. Processing is as follows:

- Declare the arrays.
- Define reference patterns for A and B.
- Initialize A and B.
- Remove the definitions of the reference patterns for A and B.
- Define new reference patterns for A and B.
- Multiply A and B, generating array C.
- Remove the definitions of the reference patterns for A and B.

The examples are presented in the following topics:

- “C/370 example”
- “COBOL example” on page 94
- “FORTRAN example” on page 98
- “Pascal example” on page 101
- “PL/I example” on page 103

C/370 example

The following example is coded in C/370:

```
#include <stdio.h>
#include <stdlib.h>
#include "csrbpc"

#define m 200
#define n 200
#define p 200
#define kelement_size 4
int chk_code(long int ret, long int reason, int linenumber);

main()
{
    long int A[m] [n];
    long int B[m] [n];
    long int C[m] [n];
    long int i;
    long int j;
    long int k;
    long int rc;
    long int rsn;
```

C/370 example

```
long int arraysize;
long int direction;
long int unitssize;
long int gap;
long int units;

arraysize = m*n*kelement_size;
direction = csr_forward;
unitssize = kelement_size*n;
gap = 0;
units = 20;

csrrip(A, &arraysize, &direction,;
      &unitssize,;
      &gap,;
      &units,;
      &rc,;
      &rsn);
chk_code(rc,rsn,__LINE__);

arraysize = m*p*kelement_size;

csrrip(B, &arraysize, &direction,;
      &unitssize,;
      &gap,;
      &units,;
      &rc,;
      &rsn);
chk_code(rc,rsn,__LINE__);
for (i=0; i<m; i++) {
  for (j=0; j<n; j++) {
    A[i][j] = i + j;
  }
}
for (i=0; i<n; i++) {
  for (j=0; j<p; j++) {
    B[i][j] = i + j;
  }
}

arraysize = m*n*kelement_size;

csrrrp(A, &arraysize,;
      &rc,;
      &rsn);
chk_code(rc,rsn,__LINE__);

arraysize = m*p*kelement_size;
csrrrp(B, &arraysize,;
      &rc,;
      &rsn);
chk_code(rc,rsn,__LINE__);

arraysize = m*n*kelement_size;
units = 25;
csrrip(A, &arraysize, &direction,;
      &unitssize,;
      &gap,;
      &units,;
      &rc,;
      &rsn);
chk_code(rc,rsn,__LINE__);

arraysize = n*p*kelement_size;
gap = (p-1)*kelement_size;
units = 50;
```

```

csrrip(B, &arraysize, &direction,;
      &unitsize,;
      &gap,;
      &units,;
      &rc,;
      &rsn);
chk_code(rc,rsn,__LINE__);
for (i=0; i<m; i++) {
  for (j=0; j<p; j++) {
    C[i][j] = 0;
    for (k=0; k<n; k++) {
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
  }
}
arraysize = m*n*kelement_size;
csrrrp(A, &arraysize,;
      &rc,;
      &rsn);
chk_code(rc,rsn,__LINE__);
arraysize = n*p*kelement_size;
csrrrp(B, &arraysize,;
      &rc,;
      &rsn);
chk_code(rc,rsn,__LINE__);
}

/* chk_code will check return code and reason code from previous */
/* calls to HLL services. It will print a message if any of the */

int chk_code(long int ret, long int reason, int linenumber)
{
  if (ret != 0)
    printf("return_code = %ld instead of 0 at line %d\n",
          ret, linenumber);
  if (reason != 0)
    printf("reason_code = %ld instead of 0 at line %d\n",
          reason, linenumber);
}

/*-----
/* JCL USED TO COMPILE, LINK, THE C/370 PROGRAM
/*-----
//CJOB JOB
//CCSTEP EXEC EDCCO,
//  CPARM='LIST,XREF,OPTIMIZE,RENT,SOURCE',
//  INFILE='REFPAT.SAMPLE.PROG(C),DISP=SHR'
//COMPILE.SYSLIN DD DSN='TEST.MPS.OBJ(C),DISP=SHR'
//COMPILE.USERLIB DD DSN=REFPAT.DECLARE.SET,DISP=SHR
//LKSTEP EXEC EDCPLO,
//  LPARM='AMOD=31,LIST,REFR,RENT,RMOD=ANY,XREF'
//PLKED.SYSIN DD DSN='TEST.MPS.OBJ(C),DISP=SHR'
//LKED.SYSLMOD DD DSN=REFPAT.USER.LOAD,DISP=SHR,
//  UNIT=3380,VOL=SER=RSMPAK
//LKED.SYSIN DD *
//  LIBRARY IN(CSRIRP,CSRRRP)
//  NAME BPGC(R)
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR
/*-----
/* JCL USED TO EXECUTE THE C/370 PROGRAM
/*-----
//CGO JOB TIME=1440,MSGLEVEL=(1,1),MSGCLASS=A
//RUN EXEC PGM=BPGC,TIME=1440
//STEPLIB DD DSN=REFPAT.USER.LOAD,DISP=SHR,
// UNIT=3380,VOL=SER=VM2TSO

```

```
//          DD   DSN=CEE.SCEERUN,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//PLIDUMP  DD   SYSOUT=*
//SYSUDUMP DD   SYSOUT=*
```

0111002

COBOL example

```
/*-----
/* THE FOLLOWING EXAMPLE IS CODED IN COBOL:
/*-----

IDENTIFICATION DIVISION.
*****
* MULTIPLY ARRAY A TIMES ARRAY B GIVING ARRAY C *
* USE THE REFERENCE PATTERN CALLABLE SERVICES TO IMPROVE THE *
* PERFORMANCE. *
*****

PROGRAM-ID. TESTCOB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

* COPY THE INCLUDE FILE (WHICH DEFINES CSRFORWARD, CSRBACKWARD)
COPY CSRPCOB.

* DIMENSIONS OF ARRAYS - A IS M BY N, B IS N BY P, C IS M BY P
1 M PIC 9(9) COMP VALUE 200.
1 N PIC 9(9) COMP VALUE 200.
1 P PIC 9(9) COMP VALUE 200.

* ARRAY DECLARATIONS FOR ARRAY A - M = 200, N = 200
1 A1.
2 A2 OCCURS 200 TIMES.
3 A3 OCCURS 200 TIMES.
4 ARRAY-A PIC S9(8).

* ARRAY DECLARATIONS FOR ARRAY B - N = 200, P = 200
1 B1.
2 B2 OCCURS 200 TIMES.
3 B3 OCCURS 200 TIMES.
4 ARRAY-B PIC S9(8).

* ARRAY DECLARATIONS FOR ARRAY C - M = 200, P = 200
1 C1.
2 C2 OCCURS 200 TIMES.
3 C3 OCCURS 200 TIMES.
4 ARRAY-C PIC S9(8).

1 I PIC 9(9) COMP.
1 J PIC 9(9) COMP.
1 K PIC 9(9) COMP.
1 X PIC 9(9) COMP.
1 ARRAY-A-SIZE PIC 9(9) COMP.
1 ARRAY-B-SIZE PIC 9(9) COMP.
1 UNITSIZE PIC 9(9) COMP.
1 GAP PIC 9(9) COMP.
1 UNITS PIC 9(9) COMP.
1 RETCODE PIC 9(9) COMP.
1 RSNCODE PIC 9(9) COMP.
PROCEDURE DIVISION.
    DISPLAY " BPAGE PROGRAM START "

* CALCULATE CSRIRP PARAMETERS FOR INITIALIZING ARRAY A
* UNITSIZE WILL BE THE SIZE OF ONE ROW.
* UNITS WILL BE 25
* SO WE'RE ASKING FOR 25 ROWS TO COME IN AT A TIME
```

```

COMPUTE ARRAY-A-SIZE = M * N * 4
COMPUTE UNITSIZE = N * 4
COMPUTE GAP = 0
COMPUTE UNITS = 25

CALL "CSRIRP" USING
  ARRAY-A(1, 1),
  ARRAY-A-SIZE,
  CSRFORWARD,
  UNITSIZE,
  GAP,
  UNITS,
  RETCODE,
  RSNCODE

DISPLAY "FIRST RETURN CODE IS "
DISPLAY RETCODE

* CALCULATE CSRIRP PARAMETERS FOR INITIALIZING ARRAY B
* UNITSIZE WILL BE THE SIZE OF ONE ROW.
* UNITS WILL BE 25
* SO WE'RE ASKING FOR 25 ROWS TO COME IN AT A TIME

COMPUTE ARRAY-B-SIZE = N * P * 4
COMPUTE UNITSIZE = P * 4
COMPUTE GAP = 0
COMPUTE UNITS = 25
CALL "CSRIRP" USING

  ARRAY-B(1, 1),
  ARRAY-B-SIZE,
  CSRFORWARD,
  UNITSIZE,
  GAP,
  UNITS,
  RETCODE,
  RSNCODE

DISPLAY "SECOND RETURN CODE IS "
DISPLAY RETCODE

* INITIALIZE EACH ARRAY A ELEMENT TO THE SUM OF ITS INDICES
PERFORM VARYING I FROM 1 BY 1 UNTIL I = M
  PERFORM VARYING J FROM 1 BY 1 UNTIL J = N
    COMPUTE X = I + J
    MOVE X TO ARRAY-A(I, J)
  END-PERFORM
END-PERFORM

* INITIALIZE EACH ARRAY B ELEMENT TO THE SUM OF ITS INDICES
PERFORM VARYING I FROM 1 BY 1 UNTIL I = N
  PERFORM VARYING J FROM 1 BY 1 UNTIL J = P
    COMPUTE X = I + J
    MOVE X TO ARRAY-B(I, J)
  END-PERFORM
END-PERFORM

* REMOVE THE REFERENCE PATTERN ESTABLISHED FOR ARRAY A
CALL "CSRIRP" USING
  ARRAY-A(1, 1),
  ARRAY-A-SIZE,
  RETCODE,
  RSNCODE

DISPLAY "THIRD RETURN CODE IS "
DISPLAY RETCODE

```

COBOL example

```
* REMOVE THE REFERENCE PATTERN ESTABLISHED FOR ARRAY B
  CALL "CSRIRP" USING
    ARRAY-B(1, 1),
    ARRAY-B-SIZE,
    RETCODE,
    RSNCODE

  DISPLAY "FOURTH RETURN CODE IS "
  DISPLAY RETCODE

* CALCULATE CSRIRP PARAMETERS FOR ARRAY A
* UNITSIZE WILL BE THE SIZE OF ONE ROW.
* UNITS WILL BE 20
* SO WE'RE ASKING FOR 20 ROWS TO COME IN AT A TIME
  COMPUTE ARRAY-A-SIZE = M * N * 4
  COMPUTE UNITSIZE = N * 4
  COMPUTE GAP = 0
  COMPUTE UNITS = 20

  CALL "CSRIRP" USING
    ARRAY-A(1, 1),
    ARRAY-A-SIZE,
    CSRFORWARD,
    UNITSIZE,
    GAP,
    UNITS,
    RETCODE,
    RSNCODE

  DISPLAY "FIFTH RETURN CODE IS "
  DISPLAY RETCODE

* CALCULATE CSRIRP PARAMETERS FOR ARRAY B
* UNITSIZE WILL BE THE SIZE OF ONE ELEMENT.
* GAP WILL BE (N-1)*4 (IE. THE REST OF THE ROW).
* UNITS WILL BE 50
* SO WE'RE ASKING FOR 50 ELEMENTS OF A COLUMN TO COME IN
* AT ONE TIME
  COMPUTE ARRAY-B-SIZE = N * P * 4
  COMPUTE UNITSIZE = 4
  COMPUTE GAP = (N - 1) * 4
  COMPUTE UNITS = 50

  CALL "CSRIRP" USING
    ARRAY-B(1, 1),
    ARRAY-B-SIZE,
    CSRFORWARD,
    UNITSIZE,
    GAP,
    UNITS,
    RETCODE,
    RSNCODE

  DISPLAY "SIXTH RETURN CODE IS "
  DISPLAY RETCODE

* MULTIPLY ARRAY A TIMES ARRAY B GIVING ARRAY C
  PERFORM VARYING I FROM 1 BY 1 UNTIL I = M
    PERFORM VARYING J FROM 1 BY 1 UNTIL J = P
      COMPUTE ARRAY-C(I, J) = 0
      PERFORM VARYING K FROM 1 BY 1 UNTIL K = N
        COMPUTE X = ARRAY-C(I, J) +
          ARRAY-A(I, K) * ARRAY-B(K, J)
      END-PERFORM
    END-PERFORM
  END-PERFORM
```

```

* REMOVE THE REFERENCE PATTERN ESTABLISHED FOR ARRAY A
  CALL "CSRRRP" USING
    ARRAY-A(1, 1),
    ARRAY-A-SIZE,
    RETCODE,
    RSNCODE

    DISPLAY "SEVENTH RETURN CODE IS "
    DISPLAY RETCODE

* REMOVE THE REFERENCE PATTERN ESTABLISHED FOR ARRAY B
  CALL "CSRRRP" USING
    ARRAY-B(1, 1),
    ARRAY-B-SIZE,
    RETCODE,
    RSNCODE

    DISPLAY "EIGHTH RETURN CODE IS "
    DISPLAY RETCODE

    DISPLAY " BPAGE PROGRAM END "
    GOBACK.

/*-----
/* JCL USED TO COMPILE, LINK, THE COBOL PROGRAM
/*-----
//FCHANGC JOB 'D3113P,D31,?', 'FCHANG6-6756', CLASS=T,
//  MSGCLASS=H, NOTIFY=FCHANG, REGION=0K
//CCSTEP EXEC EDCCO,
//  CPARAM='LIST,XREF,OPTIMIZE,RENT,SOURCE',
//  INFILE='FCHANG.PUB.TEST(C)'
//COMPILE.SYSLIN DD DSN='FCHANG.MPS.OBJ(C)', DISP=SHR'
//COMPILE.USERLIB DD DSN='FCHANG.DECLARE.SET, DISP=SHR
//LKSTEP EXEC EDCPLO,
//  LPARM='AMOD=31, LIST, REFR, RENT, RMOD=ANY, XREF'
//PLKED.SYSIN DD DSN='FCHANG.MPS.OBJ(C)', DISP=SHR'
//LKED.SYSLMOD DD DSN=RSMID.FBB4417.LINKLIB, DISP=SHR,
//  UNIT=3380, VOL=SER=RSMPAK
//LKED.SYSIN DD *
//  LIBRARY IN(CSRIRP, CSRRRP)
//  NAME BPGC(R)
//LKED.IN DD DSN=FCHANG.MPS.OBJ, DISP=SHR
/*-----
/* LINK PROGRAM
/*-----
//COBOLLK JOB
//LINKEDIT EXEC PGM=IEWL,
//  PARM='MAP,XREF,LIST,LET,AC=1,SIZE=(1000K,100K)'
//SYSLIN DD DDNAME=SYSIN
//SYSLMOD DD DSN=REFPAT.USER.LOAD, DISP=OLD
//SYSLIB DD DSN=CEE.SCEELKED, DISP=SHR
//MYLIB DD DSN=REFPAT.COBOL.OBJ, DISP=SHR
//CSRLIB DD DSN=SYS1.CSSLIB, DISP=SHR
//SYSPRINT DD SYSOUT=H
//*
//SYSUT1 DD UNIT=SYSDA, SPACE=(TRK, (20, 10))
//SYSUT2 DD UNIT=SYSDA, SPACE=(TRK, (20, 10))
//SYSIN DD *
//  INCLUDE MYLIB(COBOL)
//  LIBRARY CSRLIB(CSRIRP, CSRRRP)
//  NAME COBLOAD(R)
/*
/*-----
/* JCL USED TO EXECUTE THE COBOL PROGRAM
/*-----
//COB2 JOB MSGLEVEL=(1,1), TIME=1440
//GO EXEC PGM=COBLOAD
//STEPLIB DD DSN=CEE.SCEERUN, DISP=SHR

```

COBOL example

```
//          DD DSN=REFPAT.USER.LOAD,DISP=SHR,VOL=SER=RSMPAK,      00040001
//          UNIT=3380                                           00041001
//SYSABOUT DD SYSOUT=*                                         00050000
//SYSOUT   DD SYSOUT=A                                         00051001
//SYSDBOUT DD SYSOUT=*                                         00060000
//SYSUDUMP DD SYSOUT=*                                         00070000
```

FORTRAN example

```
*****
*                                                                 *
*   This is FORTRAN.  Followed by an assembler routine         *
*   called ADDR that has to be linkedited with the object     *
*   code from this testcase, and the CSR stubs.                *
*                                                                 *
*****
@PROCESS DC(BPAGEFOR)
PROGRAM BPAGEFOR
C
C   INCLUDE 'SYS1.SAMPLIB(CSRBPFOR)'
C
C   Multiply two arrays together - testing CSRIRP, CSRIRP services
C
C
C   INTEGER M /200/
C   INTEGER N /200/
C   INTEGER P /200/
C   PARAMETER (NKELEMENT_SIZE=4)
C   INTEGER RC,RSN
C   COMMON /WINCOM/A(200,200)
C   COMMON /WINCOM/B(200,200)
C   COMMON /WINCOM/C(200,200)
C
C   Initialize the arrays
C
C   CALL CSRIRP(A(1,1),
*           M*N*NKELEMENT_SIZE,
*           CSR_FORWARD,
*           M*N*NKELEMENT_SIZE,
*           0,
*           20,
*           RC,
*           RSN)
C   CALL CSRIRP(B(1,1),
*           N*P*NKELEMENT_SIZE,
*           CSR_FORWARD,
*           N*NKELEMENT_SIZE,
*           0,
*           20,
*           RC,
*           RSN)
C   DO 102 J = 1, N
C   DO 100 I = 1, M
C       A(I,J) = I + J
100 CONTINUE
102 CONTINUE
C   DO 106 J = 1, P
C   DO 104 I = 1, N
C       B(I,J) = I + J
104 CONTINUE
106 CONTINUE
C
C   CALL CSRIRP(A(1,1),
*           M*N*NKELEMENT_SIZE,
*           RC,
*           RSN)
```

```

        CALL CSRIRP(B(1,1),
*           N*N*NKELEMENT_SIZE,
*           RC,
*           RSN)
C
C Multiply the two arrays together
C
        CALL CSRIRP (A(1,1),
*           M*N*NKELEMENT_SIZE,
*           CSR_FORWARD,
*           N*NKELEMENT_SIZE,
*           (N-1)*NKELEMENT_SIZE,
*           50,
*           RC,
*           RSN)
        CALL CSRIRP (B(1,1),
*           N*N*NKELEMENT_SIZE,
*           CSR_FORWARD,
*           NKELEMENT_SIZE*N,
*           0,
*           20,
*           RC,
*           RSN)
        DO 112 I = 1, M
        DO 110 J = 1, N
        DO 108 K = 1, P
            C(I,J) = C(I,J) + A(I,K) * B(K,J)
108 CONTINUE
110 CONTINUE
112 CONTINUE
        CALL CSRIRP (A(1,1),
*           M*N*NKELEMENT_SIZE,
*           RC,
*           RSN)
        CALL CSRIRP (B(1,1),
*           N*N*NKELEMENT_SIZE,
*           RC,
*           RSN)

        STOP
        END

***** 00010000
* 00020000
*   THIS IS THE JCL THAT COMPILES THE PROGRAM.  * 00030000
* 00020000
***** 00080000
//FORTJOB JOB 00090007
// MSGCLASS=H,RDR=R, 00110007
// MSGLEVEL=(1,1),CLASS=T 00120000
//* 00130000
//* 00140000
//* COMPILER AND LINKEDIT FOR FORTRAN 00150000
//* 00160000
//* 00170000
//* 00180000
//VSF2CL PROC FVPGM=FORTVS2,FVREGN=2100K,FVPDECK=NODECK, 00190000
// FVPOLST=NOLIST,FVPOPT=0,FVTERM='SYSOUT=A', 00200000
// PGMNAME=MAIN,PGMLIB='&&GOSET',FVLNSPC='3200,(25,6)' 00210000
//* 00220000
//* COPYRIGHT: 5668-806 00230000
//* (C) COPYRIGHT IBM CORP 1985, 1988 00240000
//* LICENSED MATERIALS - PROPERTY OF IBM 00250000
//* REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083 00260000
//* 00270000
//* STATUS: 02.03.00 (VV.RR.MM) 00280000
//* 00290000
//* PARAMETER DEFAULT-VALUE USAGE 00300000

```

FORTRAN example

```

//*          00310000
//*          FVPGM    FORTVS2          COMPILER NAME          00320000
//*          FVREGN   2100K           FORT-STEP REGION       00330000
//*          FVPDECK  NODECK          COMPILER DECK OPTION    00340000
//*          FVPOLST  NOLIST          COMPILER LIST OPTION    00350000
//*          FVPOPT   0                COMPILER OPTIMIZATION    00360000
//*          FVTERM   SYSOUT=A        FORT.SYSTERM OPERAND    00370000
//*          FVLNSPC  3200,(25,6)     FORT.SYSLIN SPACE      00380000
//*          PGMLIB   &&G0SET          LKED.SYSLMOD DSNAME     00390000
//*          PGMNAME  MAIN             LKED.SYSLMOD MEMBER NAME 00400000
//*          00410000
//FORT EXEC PGM=&FVPGM,REGION=&FVREGN,COND=(4,LT),
//          PARM='&FVPDECK,&FVPOLST,OPT(&FVPOPT)'
//          00430000
//STEPLIB DD DSN=D24PP.FORT230.VSF2COMP,DISP=SHR
//          00440000
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=3429
//          00450000
//SYSTEM DD &FVTERM
//          00460000
//SYSPUNCH DD SYSOUT=B,DCB=BLKSIZE=3440
//          00470000
//SYSLIN DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//          SPACE=(&FVLNSPC),DCB=BLKSIZE=3200
//          00490000
//LKED EXEC PGM=HEWL,REGION=768K,COND=(4,LT),
//          PARM='LET,LIST,XREF'
//          00510000
//SYSPRINT DD SYSOUT=A
//          00520000
//SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR
//          00530000
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(200,20))
//          00540000
//SYSLMOD DD DSN=&PGMLIB.(&PGMNAME),DISP=(,PASS),UNIT=SYSDA,
//          SPACE=(TRK,(10,10,1),RLSE)
//          00560000
//SYSLIN DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//          00570000
//          DD DDNAME=SYSIN
//          00580000
// PEND
//          00590000
// EXEC VSF2CL,FVTERM='SYSOUT=H',
//          PGMNAME=FORTRAN,PGMLIB='REFPAT.USER.LOAD'
//          00680008
//FORT.SYSIN DD DSN=REFPAT.SAMPLE.PROG(FORTRAN),DISP=SHR
//          00690008
//LKED.SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR
//          00700000
//LKED.SYSLMOD DD DSN=REFPAT.USER.LOAD,DISP=SHR
//          00710007
//LKED.SYSIN DD *
//          00720000
//          INCLUDE IN(CSRIRP,CSRRRP,ADDR)
//          NAME BPGFORT(R)
//          00740006
//          00750000
//* THE CSR STUBS ARE AVAILABLE IN SYS1.CSSLIB,
//* THE OBJ FOR THE ADDR ROUTINE IS IN TEST.OBJ
//*
//*          00760007
//*          00770007
//*          00780000
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR
//          00790007
//          DD DSN=REFPAT.TEST.OBJ,DISP=SHR
//          00mm0007
*****
*
* THIS IS THE JCL I USE TO EXECUTE THE PROGRAM.
*
*****
//FON01 JOB MSGLEVEL=(1,1),TIME=1440
//VSF2G PROC GOPGM=MAIN,GOREGN=100K,
//*
//*
//* EXECUTE A FORTRAN TESTCASE - CHANGE ALL CRTFONXX TO CRTFONZZ
//*
//          GOF5DD='DDNAME=SYSIN',
//          GOF6DD='SYSOUT=A',
//          GOF7DD='SYSOUT=B'
//*
//* COPYRIGHT: 5668-806
//* (C) COPYRIGHT IBM CORP 1985, 1988
//* LICENSED MATERIALS - PROPERTY OF IBM
//* REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
//*
//* STATUS: 02.03.00 (VV.RR.MM)
//*
//* PARAMETER DEFAULT-VALUE USAGE
//*
//*          00220000
//*          00230000
//*          00240000
//*          00250000
//*          00260000

```

```

/**          GOPGM    MAIN          PROGRAM NAME          00270000
/**          GOREGN   100K         GO-STEP REGION      00280000
/**          GOF5DD   DDNAME=SYSIN  GO.FT05F001 DD OPERAND 00290000
/**          GOF6DD   SYSOUT=A     GO.FT06F001 DD OPERAND 00300000
/**          GOF7DD   SYSOUT=B     GO.FT07F001 DD OPERAND 00310000
/**
/**
//GO EXEC PGM=&GOPGM,REGION=&GOREGN,COND=(4,LT) 00340000
//STEPLIB DD DSN=CEE.SCEERUN,DISP=SHR          00350004
//FT05F001 DD &GOF5DD                          00360000
//FT06F001 DD &GOF6DD                          00370000
//FT07F001 DD &GOF7DD                          00380000
// PEND                                         00390000
//GO EXEC VSF2G,GOPGM=BPGFORT,GOREGN=999K     00400004
//GO.STEPLIB DD DSN=WINDOW.D24PP.FORTLIB,DISP=SHR, 00410004
//          VOL=SER=VM2TSO,UNIT=3380           00410104
//          DD DSN=WINDOW.R40.VSF2LOAD,DISP=SHR, 00411004
//          VOL=SER=VM2TSO,UNIT=3380           00412004
//          DD DSN=REFPAT.USER.LOAD,DISP=SHR,   00420003
//          VOL=SER=VM2TSO,UNIT=3380           00430004

```

Pascal example

```

*****
*
* PASCAL example. The data object is permanent and already
* allocated. A scroll area is used.
*
*****
program BPAGEPAS;

  %include CSRBPPAS

  CONST
    m          = 250;
    n          = 250;
    p          = 250;
    kelement_size = 4;
    a_size     = m*n*kelement_size;
    b_size     = n*p*kelement_size;
    c_size     = m*p*kelement_size;

  VAR
    a          : array (.1..m, 1..n.) of integer;
    b          : array (.1..n, 1..p.) of integer;
    c          : array (.1..m, 1..p.) of integer;
    i          : integer;
    j          : integer;
    k          : integer;
    rc         : integer;
    rsn        : integer;

  BEGIN
    csrirp (a(.1,1.), a_size, csr_forward,
            kelement_size*m,
            0,
            50,
            rc,
            rsn);
    csrirp (b(.1,1.), b_size, csr_forward,
            kelement_size*n,
            0,
            20,
            rc,
            rsn);
    for i:=1 to m do
      for j:=1 to n do

```

Pascal example

```

    a(.i,j.) := i + j;
  for i:=1 to n do
    for j:=1 to p do
      b(.i,j.) := i + j;
      csrgrp (a(.1,1.), a_size,
              rc,
              rsn);
      csrgrp (b(.1,1.), b_size,
              rc,
              rsn);
/* Multiply the two arrays together */

      csrgrp (a(.1,1.), m*n*kelement_size, csr_forward,
              kelement_size*n,
              0,
              20,
              rc,
              rsn);
      csrgrp (b(.1,1.), n*p*kelement_size, csr_forward,
              (p-1)*kelement_size,
              0,
              50,
              rc,
              rsn);
  for i:=1 to m do
    for J:=1 to p do
      begin;
      c(.i,j.) := 0;
      for k:=1 to n do
        c(.i,j.) := c(.i,j.) + a(.i,k.) * b(.k,j.);
      end;

      csrgrp (a(.1,1.), m*n*kelement_size,
              rc,
              rsn);
      csrgrp (b(.1,1.), n*p*kelement_size,
              rc,
              rsn);

  END.
***** 00010000
* * 00020000
* JCL TO COMPILE AND LINKEDIT * 00030000
* * 00040000
***** 00050000
//PASCJOB JOB 00060008
//GOGO EXEC PAS22CL 00100000
//* 00110000
//* COMPILE AND LINKEDIT FOR PASCAL 00120000
//* 00130000
//* CHANGE THE MEMBER NAME ON THE NEXT LINE AND THE 00140000
//* NAME CRTPANXX(R) SIX LINES DOWN 00150000
//* 00160000
//PASC.SYSLIB DD 00161006
// DD 00162006
// DD DSN=REFPAT.DECLARE.SET(CSRBPPAS),DISP=SHR 00163008
//PASC.SYSIN DD DSN=REFPAT.SAMPLE.PROG(PASCAL),DISP=SHR 00170008
//LKED.SYSLMOD DD DSN=REFPAT.USER.LOAD,DISP=SHR,UNIT=3380, 00180008
// VOL=SER=VM2TSO 00190009
//LKED.SYSIN DD * 00200000
LIBRARY IN(CSRIRP,CSRGRP) 00210005
NAME BPGPASC(R) 00220003
/* 00230000
//* SYS1.CSSLIB IS THE SOURCE OF THE CSR STUBS 00240008
//* 00250000
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR 00260008
*****
* *

```

```

*          JCL TO EXECUTE PASCAL                      *
*                                                                 *
*****
//PASC1JOB JOB                                         00010005
//GO EXEC PAS22CL                                     00050000
//*                                                                 00050102
//* Compile and linkedit for PASCAL                   00050202
//*                                                                 00050302
//PASC.SYSIN DD DSN=WINDOW.XAMPLE.LIB(CRTPAN06),DISP=SHR 00060006
//LKED.SYSLMOD DD DSN=WINDOW.USER.LOAD,DISP=SHR,UNIT=3380, 00560000
// VOL=SER=VM2TSO                                       00570000
//LKED.SYSIN DD *                                       00580000
LIBRARY IN(CSRSCOT,CSRSAVE,CSRREFR,CSRSAVE,CSRVIEW,CSRIDAC) 00590000
NAME CRTPAN06(R)                                       00600006
/*                                                                 00610000
//* SYS1.CSSLIB is the source of the CSR stubs         00620002
//*                                                                 00650002
//LKED.IN DD DSN=SYS1.CSSLIB,DISP=SHR                 00690000
*****
*                                                                 *
*                                                                 *
*          JCL TO COMPILE AND LINKEDIT.              *
*                                                                 *
*                                                                 *
*                                                                 *
*****
***** 00010000
*                                                                 * 00020000
* JCL TO EXECUTE. THIS ONE NEEDS A DD STATEMENT FOR THE * 00030000
* PERMANENT DIV OBJECT - CSRDD1. DATASET ALREADY EXISTS. * 00040000
*                                                                 * 00060000
***** 00070000
//PASCJOB JOB MSGLEVEL=(1,1),TIME=1440                00080002
//*                                                                 00090000
//*                                                                 00100000
//* RUN A PASCAL TESTCASE - CHANGE THE NAME ON THE NEXT LINE 00110000
//*                                                                 00/20000
//*                                                                 00130000
//GO EXEC PGM=BPGPASC                                  00140000
//STEPLIB DD DSN=REFPAT.USER.LOAD,                    00150002
// DISP=SHR,UNIT=3380,                                  00190000
// VOL=SER=VM2TSO                                       00200003
//CSRDD1 DD DSN=DIV.TESTDS,DISP=SHR                   00210000
//OUTPUT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=133)        00220000
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=133)      00230000
-----

```

PL/I example

```

*****
*                                                                 *
*          PLI example                                     *
*                                                                 *
*****
BGPPLI: PROCEDURE OPTIONS(MAIN);                      00010023
                                                    00020002
%INCLUDE SYSLIB(CSRBPPLI);                            00020122
                                                    00020222
/* INITs */                                           00021013
DCL M INIT(512) FIXED BIN(31);                        00022035
DCL N INIT(512) FIXED BIN(31);                        00023035
DCL P INIT(512) FIXED BIN(31);                        00024035
                                                    00025013
/* Arrays */                                          00026013
DCL A (M,N) BIN FIXED(31);                            /* First array */ 00029113
DCL B (N,P) BIN FIXED(31);                            /* Second array */ 00029213
DCL C (M,P) BIN FIXED(31);                            /* Product of first and second */ 00029313

```

PL/I example

```

DCL KELEMENT_SIZE INIT(4) FIXED BIN(31); /* Size of an element of an array. This value is tied directly to the data type of the three arrays (ie. FIXED(31) is 4 bytes */
/* Indices */
DCL I FIXED BIN(31),
    J FIXED BIN(31),
    K FIXED BIN(31);
/* Others */
DCL RC FIXED BIN(31);
DCL RSN FIXED BIN(31);
/* Initialize the first two arrays such that each element equals the sum of the indices for that element (eg. A(4,10) = 14 */
A(4,10) = 14 */
CALL CSRIRP (A(1,1), M*N*KELEMENT_SIZE, CSR_FORWARD,
            KELEMENT_SIZE*N,
            0,
            20,
            RC,
            RSN);
CALL CSRIRP (B(1,1), N*P*KELEMENT_SIZE, CSR_FORWARD,
            KELEMENT_SIZE*P,
            0,
            20,
            RC,
            RSN);
DO I = 1 TO M;
  DO J = 1 TO N;
    A(I,J) = I + J;
  END;
END;
DO I = 1 TO N;
  DO J = 1 TO P;
    B(I,J) = I + J;
  END;
END;
CALL CSRIRP (A(1,1), M*N*KELEMENT_SIZE,
            RC,
            RSN);
CALL CSRIRP (B(1,1), N*P*KELEMENT_SIZE,
            RC,
            RSN);
/* Multiply the two arrays together */
CALL CSRIRP (A(1,1), M*N*KELEMENT_SIZE, CSR_FORWARD,
            KELEMENT_SIZE*N,
            0,
            20,
            RC,
            RSN);
CALL CSRIRP (B(1,1), N*P*KELEMENT_SIZE, CSR_FORWARD,
            KELEMENT_SIZE,
            (P-1)*KELEMENT_SIZE,
            50,
            RC,
            RSN);
DO I = 1 TO M;
  DO J = 1 TO P;
    C(I,J) = 0;
  END;
END;

```

```

DO K = 1 TO N;                                00427713
  C(I,J) = C(I,J) + A(I,K) * B(K,J);          00427813
END;                                            00427913
END;                                           00428013
END;                                           00428113
CALL CSRRRP (A(1,1), M*N*KELEMENT_SIZE,      00428213
  RC,                                          00428313
  RSN);                                       00428513
CALL CSRRRP (B(1,1), N*P*KELEMENT_SIZE,      00428613
  RC,                                          00428913
  RSN);                                       00429413
END BPGPLI;                                   00429613
                                              00429713
                                              00430513
                                              01080024
*****
*
*
*   JCL TO COMPILE AND LINKEDIT.
*
*
*
*****
//PLIJOB JOB                                00010007
//*                                          00041001
//* PL/I Compile and Linkedit              00042001
//*                                          00043001
//* Change all CRTPLNx to CRTPLNy          00044001
//*                                          00045001
//GO EXEC PLIXCL,PARM.PLI='MACRO'           00050000
//PLI.SYSLIB DD DSN=REFPAT.DECLARE.SET,DISP=SHR
//PLI.SYSIN DD DSN=REFPAT.SAMPLE.PROG(PLI),DISP=SHR 00060008
//LKED.SYSLMOD DD DSN=REFPAT.USER.LOAD,UNIT=3380,VOL=SER=RSMPAK, 00070000
// DISP=SHR                                00080000
//LKED.SYSIN DD *                           00090000
  INCLUDE IN(CSRIRP,CSRRRP)                 00100001
  NAME BPGPLI(R)                            00110008
/*                                          00120000
//*                                          00121001
//*   SYS1.CSSLIB is source of CSR stubs    00130001
//*                                          00190000
//LKED.IN   DD DSN=SYS1.CSSLIB,DISP=SHR     00200000
//PLIJOB JOB                                00010007
*****
*
*
*   JCL TO EXECUTE.
*
*
*
*****
//PLIRUN  JOB MSGLEVEL=(1,1),TIME=1440      00010000
//*                                          00011001
//* EXECUTE A PL/I TESTCASE - CHANGE NAME ON NEXT LINE 00012001
//*                                          00013001
//GO      EXEC PGM=CRTPLN3                   00020000
//STEPLIB DD DSN=REFPAT.USER.LOAD,DISP=SHR,  00030000
// UNIT=3380,VOL=SER=VM2TS0                 00040000
// DD DSN=CEE.SCEERUN,DISP=SHR              0
//SYSABEND DD SYSOUT=*                      00070000
//SYSLOUT DD SYSOUT=*                       00080000
//SYSPRINT DD SYSOUT=*                      00090000

```

PL/I example

Part 3. Global resource serialization latch manager services

Chapter 9. Using the latch manager services

To use global resource serialization latch manager services, you issue CALLs from high level language programs. Each service requires a set of parameters coded in a specific order on the CALL statement.

This topic describes the CALL statements that invoke latch manager services. Each description includes a syntax diagram, parameter descriptions, and return and reason code explanations with recommended actions. Return and reason codes are shown in hexadecimal and decimal, along with the associated equate symbol.

This topic contains the following subtopics:

- “ISGLCRT — Create a latch set” on page 110
- “ISGLOBT — Obtain a latch” on page 114
- “ISGLREL — Release a latch” on page 117
- “ISGLPRG — Purge a requestor from a latch set” on page 120
- “ISGLPBA — Purge a group of requestors from a group of latch sets” on page 122

For information about the basic function of the latch manager, how to plan to use the latch manager, and how to use the latch manager callable services, see the serialization topic in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Syntax and linkage conventions for latch manager callable services

The latch manager callable services have the following general calling syntax:

CALL *routine_name(parameters)*

Some specific calling formats for languages that can invoke the latch manager callable services are:

C *routine_name (parm1,parm2,...return_code)*

COBOL

CALL “*routine_name*” USING *parm1,parm2,...return_code*

FORTRAN

CALL *routine_name (parm1,parm2,...return_code)*

PL/I

CALL *routine_name (parm1,parm2,...return_code)*

REXX

ADDRESS LU62 “*routine_name parm1 parm2...return_code*”

IBM provides files, called interface definition files (IDFs), that define variables and values for the parameters used with latch manager services. IBM provides IDFs for some of the listed languages. See the serialization topic in *z/OS MVS Programming: Authorized Assembler Services Guide* for information about the IDFs that are available on MVS.

ISGLCRT — Create a latch set

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.

In the following description of Latch_Create, constants defined in the latch manager IDFs are followed by their numeric equivalents; you may specify either when coding calls to Latch_Create.

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

CALL statement	Parameters
CALL ISGLCRT	(number_of_latches ,latch_set_name ,create_option ,latch_set_token ,return_code)

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.

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 can not 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.

ISGLCRT callable service

- 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 7. ISGLCRT Return Codes

Return code and Equate symbol	Meaning and Action
00 (0) ISGLCRT_SUCCESS	Meaning: The Latch_Create service completed successfully. Action: None required.
04 (4) ISGLCRT_DUPLICATE_NAME	Meaning: 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. Action: 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.
10 (16) ISGLCRT_NO_STORAGE	Meaning: Environmental error. Not enough storage was available to contain the requested number of latches. The latch manager does not create a new latch set. Action: Specify a smaller value on the number_of_latches parameter.

Examples of calls to latch manager services

The following is an example of how to call all the latch manager services in C language:

```

/*****/
/* C Example */
/*****/
#pragma linkage(setup, OS)
#pragma linkage(setprob, OS)
#include <ISGLMC.H> /* Include C language IDF */

main()
{
    const int numberOfLatches = 16; /* in this example we create 16
                                   latches */
    ISGLM_LSNM_type latchSetName
        = "EXAMPLE.ONE_LATCH_SET_NAME";
        /* set up 48-byte latch set name */
}

```

```

ISGLM_LSTK_type latchSetToken; /* latch set token - output from
                                create and input to obtain,
                                release, and purge */
int returnCode = 0; /* return code from services */

const int latchNumber = 6; /* in this example we obtain latch
                             six */
ISGLM_LRID_type requestorID = "123"; /* requestor ID - output from
                                       obtain and input to purge */
int ECB = 0; /* ECB used for latch obtain
              service */
ISGLM_EADDR_type ECBaddress = &ECB; /* pointer to ECB */
ISGLM_LTK_type latchToken; /* latch token - output from
                              obtain and input to release */

union {
    double alignment; /* force double word alignment */
    ISGLM_WA_type area; /* set up work area */
} work;

setup(); /* set supervisor state PSW */

/*****
/* create a latch set with 16 latches
*****/

isglcrt(numberOfLatches
        ,latchSetName
        ,ISGLCRT_PRIVATE
        ,&latchSetToken;
        ,&returnCode);

/*****
/* obtain latch
*****/

isglobt(latchSetToken
        ,latchNumber
        ,requestorID
        ,ISGLOBT_SYNC /* suspend until granted */
        ,ISGLOBT_EXCLUSIVE /* access option (exclusive) */
        ,&ECBaddress /* required, but not used */
        ,&latchToken /* identifies request */
        ,&work.area
        ,&returnCode);

/*****
/* release latch
*****/

isglrel(latchSetToken
        ,latchToken
        ,ISGLREL_UNCOND /* ABEND if latch not owned */
        ,&workarea
        ,&returnCode);

/*****
/* purge requestor from latch set
*****/

isglprg(latchSetToken
        ,requestorID
        ,&returnCode);

setprob(); /* set problem state PSW */
}
*****

```

ISGLCRT callable service

```
* SETSUP subroutine
*****
SETSUP  CSECT
SETSUP  AMODE 31
SETSUP  RMODE ANY
        SAVE  (14,12)          save regs
        SAC   0                ensure primary mode
        LR   12,15            establish addressability
        USING SETSUP,12
        MODESET MODE=SUP      set supervisor state
        RETURN (14,12),RC=0    restore caller's regs and return
        END SETSUP
*****
* SETPROB subroutine
*****
SETPROB CSECT
SETPROB AMODE 31
SETPROB RMODE ANY
        SAVE  (14,12)          save regs
        LR   12,15            establish addressability
        USING SETPROB,12
        MODESET MODE=PROB     set problem state
        RETURN (14,12),RC=0    restore caller's regs and return
        END SETPROB
```

ISGLOBT — Obtain a latch

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

Creates 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.
- Constants defined in the latch manager IDFs 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 constant ISGLOBT_COND and its associated value, 1.

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_set_token
- return_code

Latch_Obtain uses the following parameter for temporary storage:

- work_area

CALL statement	Parameters
CALL ISGLOBT	(latch_set_token ,latch_number ,requestor_ID ,obtain_option ,access_option ,ECB_address ,latch_token ,work_area ,return_code)

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 “ISGLPRG — Purge a requestor from a latch set” on page 120.

,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.

ISGLOBT_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.

ISGLOBT callable service

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.

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 8. ISGLOBT Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLOBT_SUCCESS	Meaning: The Latch_Obtain service completed successfully. Action: None.
04 (4) ISGLOBT_CONTENTION	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.

Example

See “Examples of calls to latch manager services” on page 112 for an example of how to call Latch_Obtain in C language.

ISGLREL — Release a latch

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.
- Constants defined in the latch manager IDFs 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 constant ISGLREL_COND and its associated value, 1.

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

Latch_Release returns a value in the following parameter:

- return_code

ISGLREL callable service

Latch_Release uses the following parameter for temporary storage:

- work_area

CALL statement	Parameters
CALL ISGLREL	(latch_set_token ,latch_token ,release_option ,work_area ,return_code)

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) 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).
- 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).

,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 9. ISGLREL Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLREL_SUCCESS	<p>Meaning: The Latch_Release service completed successfully. The caller released ownership of the specified latch request.</p> <p>Action: None.</p>
04 (4) ISGLREL_NOT_OWNED_ECB_REQUEST	<p>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.</p> <p>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).</p>

ISGLREL callable service

Table 9. ISGLREL Return Codes (continued)

Return code and Equate Symbol	Meaning and Action
08 (8) ISGLREL_STILL_SUSPENDED	<p>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.</p> <p>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.</p> <p>Action:</p> <ul style="list-style-type: none">• 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	<p>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.</p> <p>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.</p> <p>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.</p>

Example

See “Examples of calls to latch manager services” on page 112 for an example of how to call Latch_Release in C language.

ISGLPRG — Purge a requestor from a latch set

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, constants defined in the latch manager IDFs are followed by their numeric equivalents; you may specify either when coding calls to Latch_Purge.

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.

CALL statement	Parameters
CALL ISGLPRG	(latch_set_token ,requestor_ID ,return_code)

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 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 10. ISGLPRG Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLPRG_SUCCESS	Meaning: The Latch_Purge service completed successfully. Action: None.

ISGLPRG callable service

Table 10. ISGLPRG Return Codes (continued)

Return code and Equate Symbol	Meaning and Action
04 (4) ISGLPRG_DAMAGE_DETECTED	<p>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 <i>correct</i> data) for the specified requestor.</p> <p>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.</p>

Example

See “Examples of calls to latch manager services” on page 112 for an example of how to call Latch_Purge in C language.

ISGLPBA — Purge a group of requestors from a group of latch sets

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.

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

Latch_Purge_by_Address_Space returns a value in the return_code parameter.

CALL statement	Parameters
CALL ISGLPBA	(latch_set_token ,requestor_ID ,requestor_ID_mask ,latch_set_name ,latch_set_name_mask ,return_code)

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.

ISGLPBA callable service

,return_code

A fullwprd 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 11. ISGLPBA Return Codes

Return code and Equate Symbol	Meaning and Action
00 (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 <i>correct</i> 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.

Part 4. Resource recovery services (RRS)

Chapter 10. Using protected resources

Many computer resources are so critical to a company's work that the integrity of these resources must be guaranteed. If changes to the data in the resources are corrupted by a hardware or software failure, human error, or a catastrophe, the computer must be able to restore the data. These critical resources are called *protected resources* or, sometimes, *recoverable resources*.

The system, when requested, can coordinate changes to one or more protected resources so that all changes are made or no changes are made. Resources that the system can protect are, for example:

- A hierarchical database
- A relational database
- A product-specific resource

Resource recovery is the protection of the resources. Resource recovery consists of the protocols and program interfaces that allow an application program to make consistent changes to multiple protected resources.

Resource recovery programs

Three programs work together to protect resources:

- Application program: The application program accesses protected resources and requests changes to the resources.
- Resource manager: A resource manager is an authorized program that controls and manages access to a resource. A resource manager provides interfaces that allow the application program to read and change a protected resource. The resource manager also takes actions that commit or back out changes to a resource it manages.

Often an application changes more than one protected resource, so that more than one resource manager is involved.

A resource manager may be an IBM product, part of an IBM product, or a product from another vendor. A resource manager can be:

- A database manager, such as DB2®
- A program, such as IMS/ESA® Transaction Manager, that accepts work from an end user or another system and manages that work

Note: The resource manager in resource recovery is different from an RTM resource manager, which is related to the operating system's recovery termination management (RTM) and runs during termination processing.

- Sync-point manager: The sync-point manager coordinates changes to protected resources, so that all changes are made or no changes are made. The z/OS sync-point manager is recoverable resource management services (RRMS). Three MVS components provide RRMS function; because resource recovery services (RRS) provides the sync-point services, most technical information uses RRS rather than RRMS.

If your resources are distributed, so that they are on multiple systems, the communication resource manager on one system will coordinate the changes. Each communication resource manager works with RRS on its system.

RRS can enable resource recovery on a single system or, with APPC/MVS, on multiple systems.

The application program, resource manager, and sync-point manager use a two-phase commit protocol to protect resources.

Two-phase commit protocol

The two-phase commit protocol is a set of actions used to make sure that an application program makes all changes to a collection of resources or makes no changes to the collection. The protocol makes sure of the all-or-nothing changes even if the system, RRS, or the resource manager fails.

The phases of the protocol are:

- Phase 1: In the first phase, each resource manager must be prepared to either commit or backout the changes. They prepare for the commit and tell RRS either YES, the change can be made, or NO, the change cannot be made.

First, RRS decides the results of the YES or NO responses from the resource managers. If the decision is YES to commit the changes, RRS hardens the decision, meaning that it stores the decision in an RRS log.

Once a commit decision is hardened, the application changes are considered committed. If there is a failure after this point, the resource manager will make the changes during restart. Before this point, a failure causes the resource manager to back out the changes during restart.

- Phase 2: In the second phase, the resource managers commit or back out the changes.

Resource recovery process

For a look at the resource recovery process, think of a person who requests an automated teller machine (ATM) to transfer money from a savings account to a checking account. The application program receives the person's input from the ATM. Each account is in a different database. Each database has its own resource manager. The sync-point manager is RRS. Figure 11 on page 129 shows how the ATM application, resource managers, and RRS work together

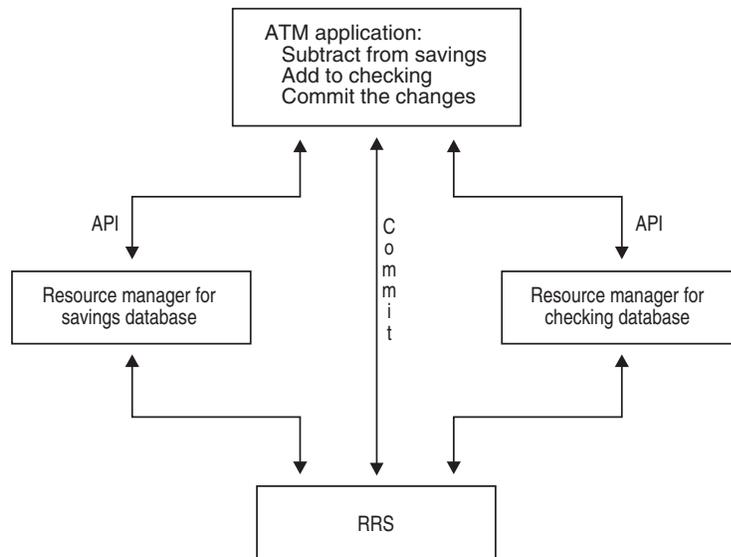


Figure 11. ATM Transaction

The actions required to process the ATM transaction are:

1. The ATM user requests transfer of money from a savings account to a checking account.
2. The ATM application program receives the ATM input.
Figure 12 on page 130 shows, for the same transaction, the sequence of the following actions, with time moving from left to right, in the two-phase commit protocol RRS uses to commit the changes. The top line in the figure shows the two phases of the protocol described in "Two-phase commit protocol" on page 128.
3. The ATM application requests the savings resource manager to subtract the money from the savings database. For this step, the application uses the resource manager's application programming interface (API).
4. The ATM application requests the checking resource manager to add the money to the checking database. The application uses this resource manager's API.
5. The ATM application issues a call to RRS to commit the database changes.
6. RRS asks the resource managers to prepare for the changes.
7. The resource managers indicate whether or not they can make the changes, by voting YES or NO. In Figure 12 on page 130, both resource managers vote YES.
8. In response, RRS notifies the resource managers to commit the changes, that is, to make the changes permanently in the databases.
9. The resource managers complete the commit and return OK to RRS.
10. RRS gives a return code to the application program, indicating that all changes were made in the databases.

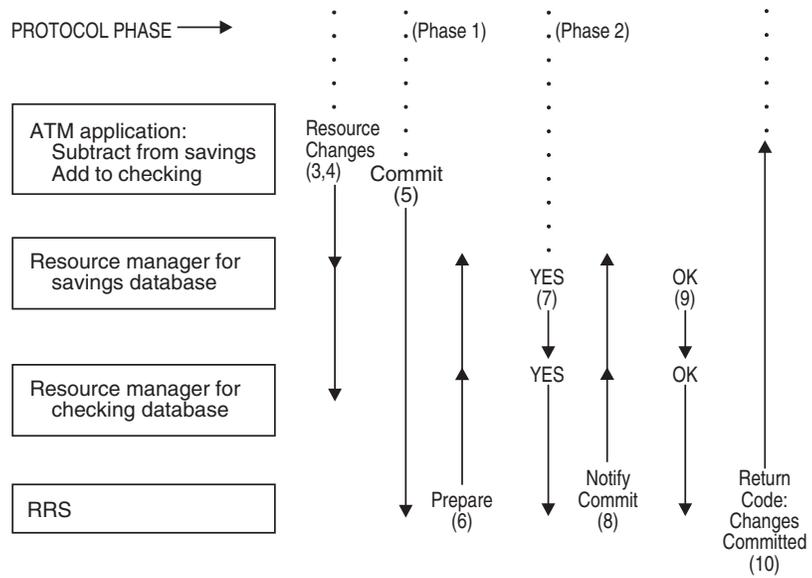


Figure 12. Two-Phase Commit Actions

If the ATM user decides not to transfer the money and presses a NO selection, the application requests backout, instead of commit, in step 6. In this case, the changes are backed out and are not actually made in any database. See Figure 13.

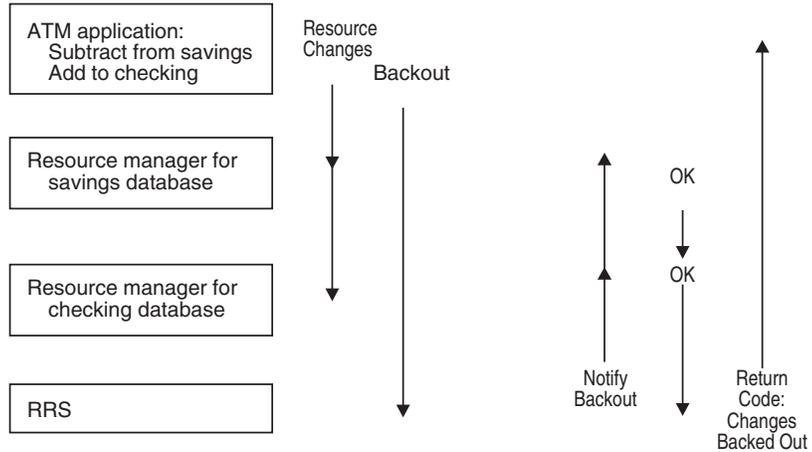


Figure 13. Backout — Application Request

Or if a resource manager cannot make the change to its database, the resource manager votes NO during prepare. If any resource manager votes NO, all of the changes are backed out. See Figure 14 on page 131.

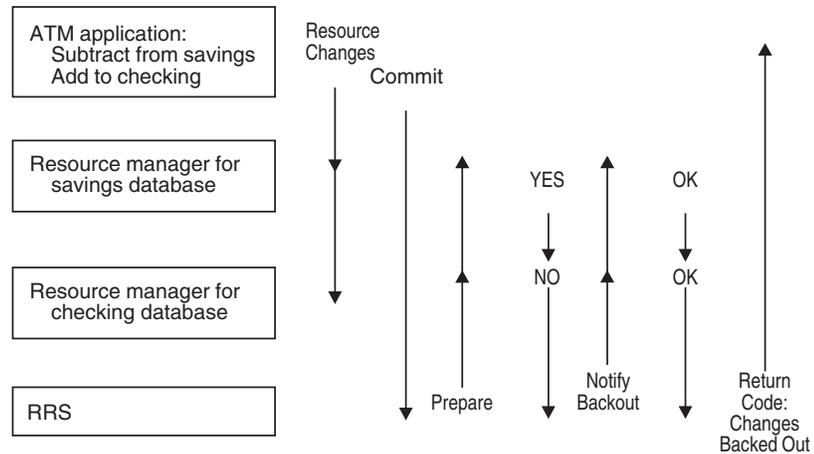


Figure 14. Backout — Resource Manager Votes NO

Requesting resource protection and recovery

To request resource protection, your application program must use resource managers that work with RRS to protect resources. The code in your application should do the following:

1. Request one or more accesses to resources for reads, writes, or both.
2. If all of the changes are to be made, request commit by issuing a call to the `Application_Commit_UR` service.
3. If none of the changes are to be made, request backout by issuing a call to the `Application_Backout_UR` service.

For details about the calls, see “`Application_Backout_UR (SRRBACK)`” on page 132 and “`Application_Commit_UR (SRRCMIT)`” on page 136.

Using distributed resource recovery

The databases for a work request may be distributed, residing on more than one system. In this case, the application program initiating the work uses a distributed communications manager, such as `APPC/MVS`, to request changes by an application program on another system. The database resource managers, communication resource managers, and RRS components work together to make or not make all changes of both application programs. Figure 15 on page 132 illustrates distributed resource recovery.

Application_Backout_UR

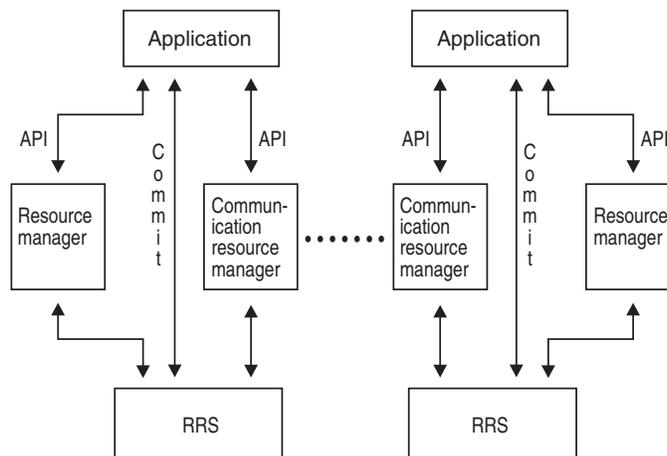


Figure 15. Transaction — Distributed Resource Recovery

Application_Backout_UR (SRRBACK)

Call the Application_Backout_UR service to indicate that the changes for the unit of recovery (UR) are not to be made. A UR represents the application's changes to resources since the last commit or backout or, for the first UR, since the beginning of the application. In response to the call, RRS requests that the resource managers return their resources to the values they had before the UR was processed.

An application might need to issue a call to the Application_Backout_UR service if:

- An APPC/MVS call returns a TAKE_BACKOUT return code. For example, a CI *send_data* call to a communications manager could return TAKE_BACKOUT.
- A resource manager call returns a return code that indicates that a resource manager directly backed out its resource. This situation can occur if the resource manager does not have the capability to return a TAKE_BACKOUT code.
- A communications resource manager call returns a return code that indicates that a backout must be done, such as a return code of COM_RESOURCE_FAILURE_NO_RETRY from a CI call.

Description

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 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 and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

The two methods described here can be used to access the callable service.

- Linkedit the stub routine ATRSCSS with the program that uses the service. ATRSCSS resides in SYS1.CSSLIB.
- Code the MVS LOAD macro within a program that uses the service to obtain the entry point address of the service. Use that address to call the service.

Additional language-specific statements may be necessary so that compilers can provide the proper assembler interface. Other programming notations, such as variable declarations, are also language-dependent.

SYS1.CSSLIB contains stubs for all of MVS's callable services including RRS. Other program products like DB2 and IMS™ also provide libraries that contain stubs for their versions of SRRBACK and SRRCMIT.

Because other program products like DB2 and IMS provide their own stubs for SRRBACK or SRRCMIT, you must make sure your program uses the correct stub. You need to take particular care when recompiling and linking any application that uses these services. When you linkedit, make sure that the data sets in the syslib concatenation are in the right order. For example, if you want a DB2 application to use the RRS callable service SRRBACK or SRRCMIT, you must ensure that SYS1.CSSLIB precedes the data sets with the stubs that DB2 provides for SRRBACK or SRRCMIT.

If you inadvertently cause your program to use SRRCMIT for RRS when it expects SRRCMIT for another program product like IMS, the application does not run correctly, and your program receives an error return code from the call to SRRCMIT.

For examples of the JCL link edit statements used with high-level languages, see Chapter 4, "Window services coding examples," on page 45 or Chapter 8, "Reference pattern services coding examples," on page 91.

High level language (HLL) definitions: The high level language (HLL) definitions for the callable service are:

HLL Definition	Description
ATRSASM	390 Assembler declarations
ATRSC	C/390 declarations
ATRSCOB	COBOL 390 declarations
ATRSPAS	Pascal 390 declarations
ATRSPLI	PL/I 390 declarations

Assembler: If you are an Assembler language caller running in AMODE 24, either use a BASSM instruction in place of the CALL or specify a LINKINST=BASSM parameter on the CALL macro. For example:

```
CALL SRRBACK(RETCODE),LINKINST=BASSM
```

COBOL: The return/reason code names and abend code names in ATRSCOB are truncated at 30 characters.

PL/I: The return/reason code names and abend code names in ATRSPLI are truncated at 31 characters.

Restrictions: The state of the UR must be **in-reset** or **in-flight**. A successful call creates a new UR that is **in-reset**.

Application_Backout_UR

The UR cannot be in local transaction mode.

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

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 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 call. If the system changes the contents of registers on which the caller depends, the caller must save them before calling the service, and restore them after the system returns control.

Performance implications: None.

Syntax: Write the call as shown in the syntax diagram. You must code the parameters in the CALL statement as shown.

CALL statement	Parameters
CALL SRRBACK	(return_code)

Parameters: The parameters are explained as follows:

return_code

Returned parameter

- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Application_Backout_UR service.

ABEND codes: The call might result in an abend X'5C4' with a reason code of X'00150000' through X'00150010'. See *z/OS MVS System Codes* for the explanations and actions.

If your application ends abnormally during sync-point processing, the condition is called an asynchronous abend, and you might need to see the programmer at your

installation responsible for managing RRS. Under information about working with application programs, *z/OS MVS Programming: Resource Recovery* contains additional details about asynchronous abends.

Issuing SETRRS CANCEL for non-resource manager programs that use the synch-point service results in an abend X'058'. When RRS restarts, transactions that were in progress are resolved.

Return codes: When the service returns control to your program, GPR 15 and *return_code* contain a hexadecimal return code, shown in the following table. If you need help with a return code, see the programmer at your installation responsible for managing RRS. Under information about working with application programs, *z/OS MVS Programming: Resource Recovery* contains additional details about these return codes.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
0	0	<p>Code: RR_OK</p> <p>Meaning: Successful completion. The resource managers returned their resources to the values they had before the UR was processed.</p> <p>Action: None.</p>
12D	301	<p>Code: RR_BACKED_OUT_OUTCOME_PENDING</p> <p>Meaning: Environmental error. The backout was not completed, for one of the following reasons:</p> <ul style="list-style-type: none"> • RRS requested that the resource managers back out the changes to the resources. However, the state of one or more of the resources is not known. • RRS is not active. • The resource manager fails with an incomplete protected interest in the UR, or RRS fails before the UR is complete. <p>Action: The action by an application depends on the system environment. Some possible actions are:</p> <ul style="list-style-type: none"> • Display a warning message to the end user. • Write an exception entry into an output log. • Abnormally end the application because the resource manager will not allow any further changes to the resource until the situation is resolved.
12E	302	<p>Code: RR_BACKED_OUT_OUTCOME_MIXED</p> <p>Meaning: Environmental error. RRS requested that the resource managers back out the changes to the resources. However, one or more resources were changed.</p> <p>Action: Same as the action for return code 12D (301).</p>

Example: In the pseudocode example, the application issues a call to request that RRS back out a UR.

Application_Backout_UR

```
⋮  
CALL SRRBACK(RETCODE)  
⋮
```

Application_Commit_UR (SRRCMIT)

Call the Application_Commit_UR service to indicate that the changes for the unit of recovery (UR) are to be made permanent. A UR represents the application's changes to resources since the last commit or backout or, for the first UR, since the beginning of the application. In response to the call, RRS requests that the resource managers make the changes permanent.

Certain resource managers, such as a communications manager, can issue a TAKE_COMMIT return code to an application that has requested changes to resources. In response to the TAKE_COMMIT code from the resource manager, the application should request the changes to the resources:

- If all of the change requests are accepted, call the Application_Commit_UR service again.
- If any of the change requests are not accepted. call the Application_Backout_UR service to back out the changes.

Description

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 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 and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

The two methods described here can be used to access the callable service.

- Linkedit the stub routine ATRSCSS with the program that uses the service. ATRSCSS resides in SYS1.CSSLIB.
- Code the MVS LOAD macro within a program that uses the service to obtain the entry point address of the service. Use that address to call the service.

Additional language-specific statements may be necessary so that compilers can provide the proper assembler interface. Other programming notations, such as variable declarations, are also language-dependent.

SYS1.CSSLIB contains stubs for all of MVS's callable services including RRS. Other program products like DB2 and IMS also provide libraries that contain stubs for their versions of SRRBACK and SRRCMIT.

Because other program products like DB2 and IMS provide their own stubs for SRRBACK or SRRCMIT, you must make sure your program uses the correct stub. You need to take particular care when recompiling and linking any application that uses these services. When you linkedit, make sure that the data sets in the syslib concatenation are in the right order. For example, if you want a DB2 application to use the RRS callable service SRRBACK or SRRCMIT, you must ensure that SYS1.CSSLIB precedes the data sets with the stubs that DB2 provides for SRRBACK or SRRCMIT.

If you inadvertently cause your program to use SRRCMIT for RRS when it expects SRRCMIT for another program product like IMS, the application does not run correctly, and your program receives an error return code from the call to SRRCMIT.

For examples of the JCL link edit statements for high-level languages, see Chapter 4, "Window services coding examples," on page 45 or Chapter 8, "Reference pattern services coding examples," on page 91.

High level language (HLL) definitions: The high level language (HLL) definitions for the callable service are:

HLL Definition	Description
ATRASM	390 Assembler declarations
ATRSC	C/390 declarations
ATRSCOB	COBOL 390 declarations
ATRSPAS	Pascal 390 declarations
ATRSPLI	PL/I 390 declarations

Assembler: If you are an Assembler language caller running in AMODE 24, either use a BASSM instruction in place of the CALL or specify a LINKINST=BASSM parameter on the CALL macro. For example:

```
CALL SRRCMIT(RETCODE),LINKINST=BASSM
```

COBOL: The return/reason code names and abend code names in ATRSCOB are truncated at 30 characters.

PL/I: The return/reason code names and abend code names in ATRSPLI are truncated at 31 characters.

Restrictions

The state of the UR that represents the changes must be **in-reset** or **in-flight**.

The UR cannot be in local transaction mode.

Input register information

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

Output register information

When control returns to the caller, the GPRs contain:

Register

Contents

0-1 Used as work registers by the system

Application_Commit_UR

- 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 call. If the system changes the contents of registers on which the caller depends, the caller must save them before calling the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown in the syntax diagram. You must code the parameter in the CALL statement as shown.

CALL statement	Parameters
CALL SRRCMIT	(return_code)

Parameters

The parameters are explained as follows:

return_code

Returned parameter

- Type: Integer
- Length: 4 bytes

Contains the return code from the Application_Commit_UR service.

ABEND codes

The call might result in an abend X'5C4' with a reason code of X'00160000' through X'00160012'. See *z/OS MVS System Codes* for the explanations and actions.

If your application ends abnormally during sync-point processing, the condition is called an asynchronous abend, and you might need to see the programmer at your installation responsible for managing RRS. Under information about working with application programs, *z/OS MVS Programming: Resource Recovery* contains additional details about asynchronous abends.

Issuing SETRRS CANCEL for non-resource manager programs that use the synch-point service results in an abend X'058'. When RRS restarts, transactions that were in progress are resolved.

Return codes

When the service returns control to your program, GPR 15 and *return_code* contain a hexadecimal return code, shown in the following table. If you need help with a return code, see the programmer at your installation responsible for managing RRS. Under information about working with application programs, *z/OS MVS Programming: Resource Recovery* contains additional details about these return codes.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
0	0	<p>Code: RR_OK</p> <p>Meaning: Successful completion. The changes to all protected resources have been made permanent.</p> <p>Action: None.</p>
65	101	<p>Code: RR_COMMITTED_OUTCOME_PENDING</p> <p>Meaning: Environmental error. The commit was not completed:</p> <ul style="list-style-type: none"> RRS requested that the resource managers make the changes to the resources permanent. However, the state of one or more of the resources is not known. <p>Action: The action by an application depends on the system environment. Some possible actions are:</p> <ul style="list-style-type: none"> Display a warning message to the end user. Write an exception entry into an output log. Abnormally end the application because the resource manager will not allow any further changes to the resource until the situation is resolved.
66	102	<p>Code: RR_COMMITTED_OUTCOME_MIXED</p> <p>Meaning: Environmental error. RRS requested that the resource managers make the changes to the resources permanent. One or more resources were changed, but one or more were not changed.</p> <p>Action: Same as the action for return code 65 (101).</p>
C8	200	<p>Code: RR_PROGRAM_STATE_CHECK</p> <p>Meaning: Environmental error. The commit failed. The resource managers did not make the changes to the resources because one of the following occurred:</p> <ul style="list-style-type: none"> A resource on the same system as the application is not in the proper state for a commit. A protected conversation is not in the required state: send, send pending, defer receive, defer allocate, sync_point, sync_point send, sync_point deallocate. A protected conversation is in send state. The communications manager started sending the basic conversation logical record, but did not finish sending it. <p>Action: Initiate an action by a resource manager to get its resource to a committable state, then call Application_Commit_UR again. For example, if the application has allocated a protected conversation through APPPC/MVS, and the conversation is in receive state, the application gets this return code. It then must use APPC/MVS services to change the conversation to send state before issuing the commit request again.</p>

Application_Commit_UR

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
12C	300	Code: RR_BACKED_OUT Meaning: Environmental error. The commit failed. The resource managers backed out the changes, returning the resources to the values they had before the UR was processed. Action: Same as the action for return code 65 (101).
12D	301	Code: RR_BACKED_OUT_OUTCOME_PENDING Meaning: Environmental error. The commit failed for one of the following reasons: <ul style="list-style-type: none">• RRS requested that the resource managers back out the changes to the resources. However, the state of one or more of the resources is not known.• RRS is not active. Action: Same as the action for return code 65 (101).
12E	302	Code: RR_BACKED_OUT_OUTCOME_MIXED Meaning: Environmental error. The commit failed. RRS requested that the resource managers back out the changes to the resources. One or more resources were backed out, but one or more were changed. Action: Same as the action for return code 65 (101).

Example

In the pseudocode example, the application issues a call to request that RRS commit a UR.

```
⋮  
CALL SRRCMIT(RETCODE)  
⋮
```

Additional callable services

Additional callable services that an authorized resource manager can use to request resource recovery services can be found in *z/OS MVS Programming: Resource Recovery*.

Part 5. CEA TSO/E address space services

Chapter 11. Introduction to CEA TSO/E address space services

The *z/OS CEA TSO/E address space manager* provides services to programmatically start and manage TSO/E address spaces and provides a communications mechanism for use between the caller and the programs running in these managed address spaces.

CEA TSO/E address space services allow callers to:

- Start a new TSO/E address space.
- End a TSO/E address space started by CEA.
- Send an attention interrupt to a TSO/E address space started by CEA.
- Obtain information about a TSO/E address space started by CEA.
- Obtain information about all the TSO/E address spaces that CEA started for an application.
- Ping a TSO/E address space that was started by CEA to prevent the address space from ending because it has been idle too long.

CEA TSO/E address space manager components

The CEA TSO/E address space manager ships with the common event adapter (CEA) component of z/OS. The CEA component provides the framework and manages the resources for the TSO/E address spaces started using the CEA TSO/E address space manager. Table 12 describes the components included in the CEA TSO/E address space manager.

Table 12. CEA TSO/E address space manager components

Component	Description
CEA address space	<p>The CEA TSO/E address space manager is integrated into the CEA address space infrastructure. The function is started automatically when CEA is started.</p> <p>Attention: If the CEA address space ends, all the TSO/E sessions created by CEA will also end. Callers will not be notified that the CEA address space has ended. Instead, when a caller attempts to invoke the CEA TSO/E address space services or use the z/OS UNIX message queue, the request will fail.</p>
Session table	<p>When the CEA TSO/E address space manager starts a new TSO/E address space, the attributes of the address space and the resources obtained are stored in an internal session table. The entry exists for the life of the session and is removed when the TSO/E address space ends.</p> <p>To display the contents of the session table, use the MODIFY CEA,DIAG,SESSTABLE command. For more details about the command, see the topic about displaying the CEA TSO/E address space information in <i>z/OS MVS System Commands</i>.</p>

Table 12. CEA TSO/E address space manager components (continued)

Component	Description
z/OS UNIX message queue	The CEA TSO/E address space manager creates and manages a z/OS UNIX message queue, which is used to facilitate communication between the caller and the TSO/E address space. For more information about the z/OS UNIX message queue, see “Communicating with programs running in the TSO/E address spaces” on page 146.
CEATsoRequest API	The CEA TSO/E address space manager provides the CEATsoRequest API, which is a 64-bit C-language based API that callers can use to request TSO/E address space services. For more information about the API, see Chapter 12, “Using CEA TSO/E address space services,” on page 151.

System prerequisites for the CEA TSO/E address space services

Table 13 describes the system prerequisites for using the CEA TSO/E address space services.

Table 13. System prerequisites

Prerequisite	Description
CEA must be active.	<p>The CEA TSO/E address space manager runs in the CEA address space, which is started automatically during z/OS initialization. If your installation has stopped CEA, restart it. Otherwise, the services are not enabled.</p> <p>To determine whether the CEA address space is active, enter the following z/OS system console command:</p> <pre>D A,CEA</pre>
The TRUSTED attribute must be assigned to the CEA started task.	<p>To allow the CEA TSO/E address space manager to access or create any resource it needs, the CEA started task requires the TRUSTED(YES) attribute to be set on the RDEFINE STARTED CEA.** definition.</p> <p>If the TRUSTED attribute is not assigned to the CEA started task, the CEA TSO/E address space manager services might not be operational. For example, the services will not be able to create or access z/OS UNIX message queues.</p> <p>For more information about the RACF® TRUSTED attribute, see the topic on associating started procedures and jobs with user IDs in <i>z/OS Security Server RACF System Programmer's Guide</i>, and the topic on using started procedures in <i>z/OS Security Server RACF Security Administrator's Guide</i>.</p>
The CEA address space must be started in full function mode.	<p>Because the CEATsoRequest API requires z/OS UNIX System Services, CEA must be started in full function mode. For information about starting CEA in full function mode, see the topic about customizing CEA in <i>z/OS Planning for Installation</i>.</p>
Callers must be authorized to SAF resource profile CEA.CEATSO.TSOREQUEST.	<p>To access the CEATsoRequest API, callers must be authorized by their security product to SAF resource profile CEA.CEATSO.TSOREQUEST.</p>

Table 13. System prerequisites (continued)

Prerequisite	Description
Users must be authorized to the appropriate resources.	The user ID of the user for whom the caller is requesting TSO/E address space services must be authorized to use TSO/E, OMVS, and any other resources the address space requires.

Working with TSO/E address spaces started by CEA

By default, the CEA TSO/E address space manager can create a maximum of 10 concurrent address spaces for a single user, and can create a maximum of 50 concurrent TSO/E address spaces. These settings are configurable through the MAXSESSIONS and MAXSESSPERUSER operands provided for the TSOASMGR statement in the CEAPRMxx parmlib member. For more information, see *z/OS MVS Initialization and Tuning Reference*.

You can use the same processes that you use to work with other TSO/E address spaces when working with the TSO/E address spaces that are created by the CEA TSO/E address space manager. For example, you can issue the `D TS z/OS` console command to display information about TSO/E address spaces, or you can issue the `C u=userid,A=asid` console command to cancel a TSO/E address space. For the display command, the TSO/E address spaces will appear in the list, indistinguishable from the other TSO/E address spaces. Note that TSO/E sessions started by CEA do not add to the count for the total maximum sessions for VTAM®.

You can also display information about these TSO/E address spaces using SDSF, a REXX EXEC, or a CLIST. Note that the application identifier that was specified when the TSO/E session was started is displayed where you would typically expect to see a terminal ID.

For example, if the CEA TSO/E address space manager starts a TSO/E session for the z/OSMF ISPF task, which has an application identifier equal to IZUIS, and you issue the REXX EXEC depicted in Figure 16, you will obtain the results depicted in Figure 17 on page 146:

```
/* REXX */
trace all
myapp = sysvar('systemid')
say myapp
exit 0
```

Figure 16. Sample REXX EXEC

```

TSO Messages - ASID: 0x38

READY
COMMAND REXX NOT FOUND
DATA SET CEAID.CEA.REXX NOT IN CATALOG OR CATALOG CANNOT BE ACCESSED
COMMAND CAT NOT FOUND

-----
3 *-* myapp = sysvar('systemid')
>>> "IZUIS"
4 *-* say myapp
>>> "IZUIS"

IZUIS
5 *-* exit 0
>>> "0"

[OK] [Attention] [Clear] [Help]

```

Figure 17. Example illustrating that the REXX SYSTERMID is the same as the z/OSMF ISPF application identifier

Communicating with programs running in the TSO/E address spaces

A z/OS UNIX message queue is the mechanism the CEA TSO/E address space manager uses for allowing communications between the caller and TSO/E, ISPF, and other programs running in the TSO/E address space. To communicate with the TSO/E address space, callers must read data from and write data to the message queue.

The CEA TSO/E address space manager creates a z/OS UNIX message queue for each TSO/E address space when the TSO/E address space is started, and anchors the message queue in the session table for the duration of the session. The CEA TSO/E address space manager deletes the message queue when the TSO/E address space ends.

Messages that typically are written to a 3270-type terminal are translated to UTF-8, converted to a JSON format, and written to the z/OS UNIX message queue along with identifying header information and a message type identifier. For a list of the message type identifiers, see Table 14.

Table 14. Message type identifiers

Message Type ID	Description
1	Control data for the client.
2	TSO/E data for the client.
3	ISPF data for the client.
4 thru 32768	Reserved for IBM.
32769	Control TSO/E data from the client.
32770	TSO/E data from the client.
32771	ISPF data from the client.

Table 14. Message type identifiers (continued)

Message Type ID	Description
32772 thru 65535	Reserved for IBM.
65536 and above	Available for use by applications.

For information about the JSON format used for TSO/E messages, see “JSON format for TSO/E messages.” For the JSON format used for ISPF messages, see the topic about JSON data structures and variables used to communicate between ISPF and a client in *z/OS V2R2 ISPF Services Guide*.

JSON format for TSO/E messages

TSO/E messages are written to the z/OS UNIX message queue using message type identifiers 2 and 32770 and are formatted as follows:

```
{"message-type":{"VERSION":"JSON-version","data-type":"data-value"}}
```

where:

message-type

Keyword that identifies the type of TSO/E message. Table 15 lists and describes the message types that can be used for message type identifiers 2 and 32770.

Table 15. Message types

Message Type	Description	Message Type ID
TSO MESSAGE	Indicates that the system has created data or a message to be displayed on the client. The caller should read the message and display it accordingly.	2
TSO PROMPT	Indicates that the system requires a response from the client.	2
TSO RESPONSE	Indicates that a response was created by the client in response to a prompt. Callers should use this keyword when writing a response to the message queue.	32770

JSON-version

A four-digit number that identifies the JSON version used to format the message.

data-type

Keyword that describes the type of data included in the *data-value* variable. Table 16 lists and describes the data types that can be used for each TSO/E message type.

Table 16. Data types

Data Type	Description	Message Type
DATA	Indicates that the data included in the <i>data-value</i> variable is either a message from the system or a response from the client. For this data type, the <i>data-value</i> variable is a character string that can contain up to 32,767 bytes.	TSO MESSAGE and TSO RESPONSE

Table 16. Data types (continued)

Data Type	Description	Message Type
HIDDEN	Indicates whether the client should hide or mask the response. For this data type, the <i>data-value</i> variable is a Boolean that can have the value of either TRUE or FALSE. When TRUE, this tells the client to hide or mask the response as it is entered. Otherwise, the response will display as it is entered.	TSO PROMPT
ACTION	Indicates that the caller would like to interrupt or end a process that is in progress. For this data type, specify ATTN as the value for the <i>data-value</i> variable. Callers should use the CEATsoRequest API to issue the CeaTsoAttn request type before using a message to issue an attention interrupt. Use this data type only if the CeaTsoAttn request fails.	TSO RESPONSE

Sample TSO/E messages written to the z/OS UNIX message queue

Figure 18 provides an example that illustrates how TSO/E messages appear on the z/OS UNIX message queue.

Note: The message type identifiers are not part of the JSON structure. They are included for illustration purposes only.

```

2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"IKJ56455I
IBMUSER LOGON IN PROGRESS AT 03:46:24 ON OCTOBER 12, 2011"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"IKJ56951I NO BROADCAST MESSAGES
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"READY "}}
2 {"TSO PROMPT":{"VERSION":"0100","HIDDEN":"FALSE"}}
32770 {"TSO RESPONSE":{"VERSION":"0100","DATA":"TIME"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"IKJ56650I TIME-03:46:50 AM.
CPU-00:00:00 SERVICE-775140 SESSION-00:00:26 OCTOBER 12,2011"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"READY "}}
2 {"TSO PROMPT":{"VERSION":"0100","HIDDEN":"FALSE"}}
32770 {"TSO RESPONSE":{"VERSION":"0100","DATA":"ALLOC DA"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"IKJ56700A ENTER DATA SET NAME
OR * -
"}}
2 {"TSO PROMPT":{"VERSION":"0100","HIDDEN":"FALSE"}}
32770 {"TSO RESPONSE":{"VERSION":"0100","DATA":"'sys1.brodcast'"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"IKJ56225I DATA SET SYS1.BROADCAST
ALREADY IN USE, TRY LATER+"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"IKJ56225I DATA SET IS ALLOCATED
TO ANOTHER JOB OR USER"}}
2 {"TSO MESSAGE":{"VERSION":"0100","DATA":"READY "}}
2 {"TSO PROMPT":{"VERSION":"0100","HIDDEN":"FALSE"}}
32770 {"TSO RESPONSE":{"VERSION":"0100","DATA":"LOGOFF"}}

```

Figure 18. Sample TSO/E messages written to the queue

Reconnecting to CEA TSO/E address spaces

When a user requests to end a TSO/E session created by CEA, if the caller has not set the abnormal logoff flag (CEATSO_ABLOGOFF) or the no reconnect flag (CEATSO_NORECONN), the CEA TSO/E address space manager can intercept that request and place the session in a dormant state instead of ending it.

A *dormant TSO/E session* is a session that has been deactivated for communication through its message queue but remains available at a TSO/E READY prompt for a period of time so that the user can reconnect to it. Reconnecting to a dormant session is faster and uses fewer resources than constructing a new session because the session resources are retained and reused when the user reconnects to the session.

To enable the CEA reconnect feature, which is disabled by default, specify non-zero values for the RECONSESSIONS and RECONTIME statements in the TSOASMGR parmlib statement in the CEAPRMxx parmlib member. The RECONSESSIONS statement indicates how many dormant sessions can be created for each user, and the RECONTIME statement indicates the amount of time a dormant session remains a candidate for reconnection.

The CEA TSO/E address space manager can create a maximum of three dormant sessions per user and can keep a dormant session available for reconnection for a maximum of 23 hours, 59 minutes, and 59 seconds. The settings you specify for the TSOASMGR parmlib statement affect all of the TSO/E sessions that are managed by the CEA TSO/E address space manager. For more information about the TSOASMGR parmlib statement, see the topic about the CEAPRMxx parmlib member in *z/OS MVS Initialization and Tuning Reference*.

When the CEA reconnect feature is enabled, to reconnect to a dormant session, the user must do the following:

- Request to start a new TSO/E session before the specified RECONTIME expires. After the RECONTIME expires, the session remains in a dormant state until CEA ends it; however, the session is no longer a candidate for reconnection.
- Use the same security credentials and logon parameters that were used for the dormant session.

If no dormant sessions are available that satisfy these requirements, the CEA TSO/E address space manager will create a new address space for the user.

Dormant TSO/E sessions do not interfere with the maximum number of sessions allowed. That is, if a user tries to create a new session and the number of active and dormant sessions equal the maximum allowed, the CEA TSO/E address space manager will end a dormant session and create a new session for the user.

Idle time versus RECONTIME

Each dormant TSO/E session has an idle application time, which is not adjustable, and a reconnect time (RECONTIME). The idle time cannot exceed 15 minutes. Otherwise, the CEA TSO/E address space manager will end the session regardless of reconnect time. To prevent your dormant sessions from ending because of idle time, issue a ping request at least once every 15 minutes, which informs CEA that all of the sessions for your application are still active. For more information, see “CeaTsoPing - Sending a ping on behalf of an application” on page 160.

TSO/E LOGON RECONNECT operand versus CEA reconnect

The TSO/E LOGON command is not supported for CEA-managed TSO/E sessions, and the capability provided by the TSO/E LOGON RECONNECT operand is different from the CEA reconnect feature. For more information about the TSO/E LOGON RECONNECT operand, see the topic about LOGON command operands in *z/OS TSO/E Command Reference*.

Chapter 12. Using CEA TSO/E address space services

To use CEA TSO/E address space services, you issue CALLs from high-level language programs that invoke the CEATsoRequest API. The API is a 64-bit C-language based interface that the CEA TSO/E address space manager uses to receive requests from callers and to determine what action to take to process the request.

The CEATsoRequest API supports the following request types:

- **CeaTsoStart.** Start a TSO/E address space.
- **CeaTsoAttn.** Send an attention interrupt to a TSO/E address space started by CEA.
- **CeaTsoEnd.** End a TSO/E address space started by CEA.
- **CeaTsoPing.** Ping a TSO/E address space that was started by CEA to prevent the address space from ending because it has been idle too long.
- **CeaTsoQuery.** Obtain information about a specific TSO/E address space started by CEA.
- **CeaTsoQueryApp.** Obtain information about all the TSO/E address spaces that CEA started for an application.

For more details about the request types, see “Understanding the request types” on page 157.

Invoking the CEATsoRequest API

The format to use to call the CEATsoRequest API follows:

```
#include <ceaytsor.h>
#include <ceaxrdef.h>

int32_t CEATsoRequest(CEATsoRequestStruct_t* RequestStruct,
                    CEATsoQueryStruct_t* QueryStruct,
                    CEATsoError_t* ErrorStruct)
```

The call format is the same for each request type. The only difference is the fields that are required for each structure. For a description of each parameter and all the possible fields that can be included in each structure, see “Parameters” on page 152. For a list of the fields that are required for each request type, see “Understanding the request types” on page 157.

The CEATsoRequest API is used as a dynamically loaded library. The file ceasapit.x, which exists in /usr/lib, contains the sidedeck needed to link your program to the DLL. The contents of the file are depicted in Figure 19.

```
IMPORT CODE64, 'ceasapit.dll', 'CEATsoRequest'
```

Figure 19. Contents included in the ceasapit.x file

To compile your programs, the following header files are required: ceaytsor.h and ceaxrdef.h. The header files are stored in partitioned data set SYS1.SIEAHDRV. The contents of the header files are provided in “CEAYTSOR header file” on page 170 and “CEAXRDEF header file” on page 174.

Parameters

RequestStruct

Pointer to the CEATsoRequestStruct structure. The layout of the CEATsoRequestStruct structure follows:

```
struct CEATsoRequestStruct_s {
    char          ceatso_eyecatcher[8];
    uint32_t      ceatso_version;
    uint32_t      ceatso_requesttype;
    char          ceatso_userid[8];
    uint32_t      ceatso_asid;
    char          ceatso_logonproc[8];
    char          ceatso_command[80];
    uint16_t      ceatso_numqueryreq;
    uint16_t      ceatso_numqueryrslt;
    uint32_t      ceatso_duration;
    uint32_t      ceatso_msgqueueid;
    uint16_t      ceatso_charset;
    uint16_t      ceatso_codepage;
    uint16_t      ceatso_screenrows;
    uint16_t      ceatso_screencols;
    char          ceatso_account[40];
    char          ceatso_group[8];
    char          ceatso_region[7];
    char          ceatso_instance[1];
    char          ceatso_apptag[8];
    char          ceatso_stoken[8];
    uint32_t      ceatso_ascbaddr;
    uint16_t      ceatso_flags;
    uint16_t      ceatso_index;
    char          rsvd1[8];
};
typedef struct CEATsoRequestStruct_s CEATsoRequestStruct_t;
```

The fields in the CEATsoRequestStruct structure are explained as follows:

ceatso_eyecatcher

Eye catcher. Specify 'CEAYTSOR'.

ceatso_version

Structure version number.

ceatso_requesttype

Type of request. Specify one of the following values:

- CeaTsoStart
- CeaTsoAttn
- CeaTsoEnd
- CeaTsoPing
- CeaTsoQuery
- CeaTsoQueryApp

For more details about each request type, see "Understanding the request types" on page 157.

ceatso_userid

User ID of the authenticated user for which the TSO/E address space was created.

ceatso_asid

The address space ID (ASID) for the TSO/E address space.

ceatso_logonproc

Name of the TSO/E logon procedure to use to log onto the TSO/E address space.

ceatso_command

Unused.

ceatso_numqueryreq

Maximum number of sessions to query.

ceatso_numqueryrslt

Number of sessions found that satisfy the query.

ceatso_duration

Unused.

ceatso_msgqueueid

The ID of the z/OS UNIX message queue that is used for communications between the caller and the TSO/E session.

ceatso_charset

Character set to use for the caller's TSO/E address space. This value is used by the applications running in the TSO/E address space to convert messages and responses from UTF-8 to EBCDIC. The default character set, which is 697 decimal, will be used if zero is specified as the value.

ceatso_codepage

Codepage to use for the caller's TSO/E address space. This value is used by the applications running in the TSO/E address space to convert messages and responses from UTF-8 to EBCDIC. The default codepage, which is 1047 decimal, will be used if zero is specified as the value.

ceatso_screenrows

Number of rows to be displayed on the screen. The default number of rows, which is 24, will be used if zero is specified as the value.

ceatso_screencols

Number of columns to be displayed on the screen. The default number of columns, which is 80, will be used if zero is specified as the value.

ceatso_account

TSO/E account number.

ceatso_group

TSO/E group name.

ceatso_region

Region size used for the TSO/E address space.

ceatso_instance

Number of active TSO/E address spaces that were started by CEA for the corresponding user ID. In the session table, this value is stored with the oldest TSO/E session entry created for the user.

ceatso_apptag

Identifies the application that is responsible for creating the TSO/E address space.

ceatso_stoken

A token that uniquely identifies the TSO/E address space.

ceatso_ascbaddr

Address of the address space control block that was created for the TSO/E address space.

ceatso_flags

When ending a TSO/E session, you can set the following flags:

- **CEATSO_ABLOGOFF (0x8000)**. If this flag is set, the CANCEL command will be issued to end the TSO/E session regardless of whether the CEA reconnect feature is enabled. Otherwise, the LOGOFF command will be issued or the TSO/E session will be placed in a dormant state as a candidate for reconnection.
- **CEATSO_NORECONN (0x4000)**. If this flag is set, the CEA TSO/E address space manager will end the TSO/E session even if the CEA reconnect feature is enabled. That is, if the client allows users to set this flag, users can force the CEA TSO/E address space manager to end a TSO/E session even if your installation has enabled the reconnect feature. For more information about the reconnect feature, see “Reconnecting to CEA TSO/E address spaces” on page 148.

When starting a TSO/E session, the CEA TSO/E address space manager sets the CEATSO_RECONNECTD (0x2000) flag if the user was connected to a dormant TSO/E session instead of a new session.

ceatso_index

The index value, STOKEN, and ASID together identify the TSO/E address space to the CEA TSO/E address space services.

rsvd1 Reserved for future use.

QueryStruct

Pointer to the CEATsoQueryStruct structure. This structure is used to return query results for the CeaTsoQuery and CeaTsoQueryApp request types. The layout of the CEATsoQueryStruct structure follows:

```
struct CEATsoQueryStruct_s{
    char          ceatsoq_eyecatcher[8];
    uint32_t      ceatsoq_version;
    uint32_t      ceatsoq_requesttype;
    char          ceatsoq_userid[8];
    uint32_t      ceatsoq_asid;
    char          ceatsoq_logonproc[8];
    char          ceatsoq_command[80];
    uint16_t      ceatsoq_numqueryreq;
    uint16_t      ceatsoq_numqueryrslt;
    uint32_t      ceatsoq_duration;
    uint32_t      ceatsoq_msgqueueid;
    uint16_t      ceatsoq_charset;
    uint16_t      ceatsoq_codepage;
    uint16_t      ceatsoq_screenrows;
    uint16_t      ceatsoq_screencols;
    char          ceatsoq_account[40];
    char          ceatsoq_group[8];
    char          ceatsoq_region[7];
    char          ceatsoq_instance[1];
    char          ceatsoq_apptag[8];
    char          ceatsoq_stoken[8];
    uint32_t      ceatsoq_ascbaddr;
    uint16_t      ceatsoq_flags;
    uint16_t      ceatsoq_index;
    char          rsvd1[8];
};
typedef struct CEATsoQueryStruct_s CEATsoQueryStruct_t;
```

The fields in the CEATsoQueryStruct structure are explained as follows:

ceatso_eyecatcher

Eye catcher. The value is 'CEAYTSOQ'.

ceatso_version

Structure version number.

ceatso_requesttype

Type of request. The CeaTsoQueryStruct returns results for the CeaTsoQuery and CeaTsoQueryApp request types. For more details about each request type, see "Understanding the request types" on page 157.

ceatso_userid

User ID of the authenticated user for which the TSO/E address space was created.

ceatso_asid

The address space ID (ASID) for the TSO/E address space.

ceatso_logonproc

Name of the TSO/E logon procedure to use to log onto the TSO/E address space.

ceatso_command

Unused.

ceatso_numqueryreq

Maximum number of sessions to query.

ceatso_numqueryrslt

Number of sessions found that satisfy the query.

ceatso_duration

Unused.

ceatso_msgqueueid

The ID of the z/OS UNIX message queue that is used for communications between the caller and the TSO/E session.

ceatso_charset

Character set to use for the caller's TSO/E address space. This value is used by the applications running in the TSO/E address space to convert messages and responses from UTF-8 to EBCDIC. The default character set, which is 697 decimal, will be used if zero is specified as the value.

ceatso_codepage

Codepage to use for the caller's TSO/E address space. This value is used by the applications running in the TSO/E address space to convert messages and responses from UTF-8 to EBCDIC. The default codepage, which is 1047 decimal, will be used if zero is specified as the value.

ceatso_screenrows

Number of rows to be displayed on the screen. The default number of rows, which is 24, will be used if zero is specified as the value.

ceatso_screencols

Number of columns to be displayed on the screen. The default number of columns, which is 80, will be used if zero is specified as the value.

ceatso_account

TSO/E account number.

ceatso_group

TSO/E group name.

ceatso_region

Region size used for the TSO/E address space.

ceatso_instance

Number of active TSO/E address spaces that were started by CEA for the corresponding user ID. In the session table, this value is stored with the oldest TSO/E session entry created for the user.

ceatso_apptag

Identifies the application that is responsible for creating the TSO/E address space.

ceatso_stoken

A token that uniquely identifies the TSO/E address space.

ceatso_ascbaddr

Address of the address space control block that was created for the TSO/E address space.

ceatso_flags

When ending a TSO/E session, you can set the following flags:

- **CEATSO_ABLOGOFF (0x8000)**. If this flag is set, the CANCEL command will be issued to end the TSO/E session regardless of whether the CEA reconnect feature is enabled. Otherwise, the LOGOFF command will be issued or the TSO/E session will be placed in a dormant state as a candidate for reconnection.
- **CEATSO_NORECONN (0x4000)**. If this flag is set, the CEA TSO/E address space manager will end the TSO/E session even if the CEA reconnect feature is enabled. That is, if the client allows users to set this flag, users can force the CEA TSO/E address space manager to end a TSO/E session even if your installation has enabled the reconnect feature. For more information about the reconnect feature, see "Reconnecting to CEA TSO/E address spaces" on page 148.

When starting a TSO/E session, the CEA TSO/E address space manager sets the CEATSO_RECONNECTD (0x2000) flag if the user was connected to a dormant TSO/E session instead of a new session.

ceatso_index

The index value, STOKEN, and ASID together identify the TSO/E address space to the CEA TSO/E address space services.

rsvd1 Reserved for future use.

ErrorStruct

Pointer to the CEATsoErrorStruct structure. This structure contains information about the results of the request. The layout of the CEATsoErrorStruct structure follows:

```

struct CEATsoError_s {
    char          eyeCatcher[8];
    uint32_t      version;
    int32_t       returnCode;
    uint32_t      reasonCode;
}

```

```

    CEATsoDiag_t diag;
};
typedef struct CEATsoError_s CEATsoError_t;

```

The fields in the CEATsoErrorStruct structure are explained as follows:

eyeCatcher

Eye catcher. Specify 'CEAIERRO'.

version

Structure version number.

returnCode

Return code. For more information about return codes, see "Return codes" on page 163.

reasonCode

Reason code. For more information about reason codes, see "Reason codes" on page 163.

diag

Diagnostic codes, which are mapped by a CEATsoDiag_t structure. This structure can contain up to four diagnostic codes that provide more details about the failure. For more information about diagnostic codes, see "Diagnostic codes" on page 167.

Requirements for callers

To send requests to the API, the environment of the caller must satisfy the following requirements:

- **Minimum authorization:** Problem state
- **Dispatchable unit mode:** Task
- **Cross memory mode:** PASN=HASN=SASN
- **AMODE:** 64-bit
- **ASC mode:** Primary
- **Interrupt status:** Enabled for I/O and external interrupts
- **Locks:** No locks held
- **Linkage:** Uses standard C linkage conventions
- **Library path (LIBPATH):** Must be set to include /usr/lib

Understanding the request types

This section describes the request types that are provided by the CEATsoRequest API. For a description of the API, including the call format and parameters, see "Invoking the CEATsoRequest API" on page 151.

CeaTsoStart - Starting a new TSO/E session

Use the CeaTsoStart request type to start a new TSO/E address space or to reconnect to a dormant TSO/E session. When you start a new TSO/E address space, a z/OS UNIX message queue is also created to enable communication between the caller and the TSO/E address space. When you reconnect to a TSO/E session, the existing message queue is reused.

The TSO/E address space is started or reconnected to using the security environment of the caller. If there is task-level security, it is used for the address space. Otherwise, the address space security environment is used. The user tokens (UTOKENs) from both environments are saved and are used to verify subsequent requests.

Table 17 lists the input callers must provide for each structure used for this request type and the output that will be provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields listed for each structure, see “Parameters” on page 152.

Table 17. Input and output for each structure used for the CeaTsoStart request type

Structure	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version • ceatso_requesttype • ceatso_logonproc • ceatso_charset • ceatso_codepage • ceatso_screenrows • ceatso_screencols • ceatso_account • ceatso_group • ceatso_region • ceatso_apptag 	<p>If the return code is CEASUCCESS, the following fields are returned:</p> <ul style="list-style-type: none"> • ceatso_userid • ceatso_asid • ceatso_msgqueueid • ceatso_stoken • ceatso_index • ceatso_flags. The value is <i>tso_reconnected</i> if the CEA TSO/E address space manager connected the user to a dormant TSO/E session.
CeaTsoQueryStruct	Not used for this request type.	Not used for this request type.
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

CeaTsoAttn - Sending an attention interrupt to a TSO/E session

Use the CeaTsoAttn request type to send an attention interrupt to a TSO/E address space started by CEA. An attention interrupt allows you to interrupt or end a process that is taking place. This request type is useful if the client is stuck at a prompt or if you submitted a request to which the system is not responding.

To perform this request, the CEA TSO/E address space manager extracts the caller’s security UTOKEN from the caller’s environment and uses it when needed.

Table 18 lists the input callers must provide for each structure used for this request type and the output that will be provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields listed for each structure, see “Parameters” on page 152.

Table 18. Input and output for each structure used for the CeaTsoAttn request type

Structure	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version • ceatso_requesttype • ceatso_asid • ceatso_apptag • ceatso_stoken • ceatso_index 	None

Table 18. Input and output for each structure used for the CeaTsoAttn request type (continued)

Structure	Required Input	Output
CeaTsoQueryStruct	Not used for this request type.	Not used for this request type.
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

CeaTsoEnd - Ending a TSO/E session

Use the CeaTsoEnd request type to end a TSO/E address space started by CEA or to place the session into a dormant state. When you end a TSO/E address space, all of the associated resources are returned to the system, including the z/OS UNIX message queue that was used for communicating with the session.

If the CEA reconnect feature is enabled and the caller has not set the CEATSO_ABLOGOFF flag (0x8000) or the CEATSO_NORECONN flag (0x4000), the CEA TSO/E address space manager will intercept the CeaTsoEnd request and place the TSO/E session in a dormant state instead of ending it. In this case, some of the session resources are retained and reused when the user reconnects to the session. For more information about the reconnect feature, see “Reconnecting to CEA TSO/E address spaces” on page 148.

To perform the CeaTsoEnd request, the CEA TSO/E address space manager extracts the caller’s security UTOKEN from the caller’s environment and uses it when needed.

Table 19 lists the input callers must provide for each structure used for this request type and the output that will be provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields listed for each structure, see “Parameters” on page 152.

Table 19. Input and output for each structure used for the CeaTsoEnd request type

Structure	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version • ceatso_requesttype • ceatso_asid • ceatso_apptag • ceatso_stoken • ceatso_index Optional input: <ul style="list-style-type: none"> • ceatso_flags 	None
CeaTsoQueryStruct	Not used for this request type.	Not used for this request type.
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

CeaTsoPing - Sending a ping on behalf of an application

Each TSO/E session has an idle application time that the CEA TSO/E address space manager uses to determine if the application that is associated with the session is active. If the idle application time is 15 minutes, the application is considered to be inactive. In which case, the CEA TSO/E address space manager ends all the CEA-managed TSO/E sessions for that application that have the same application identifier.

To prevent TSO/E sessions from ending because of idle application time, callers can use the CeaTsoPing request type to issue a ping request at least once every 15 minutes. Doing so informs CEA that the application is still active, and causes the CEA TSO/E address space manager to reset the idle application time for all the CEA-managed TSO/E sessions that have the same application identifier.

To perform this request, the CEA TSO/E address space manager extracts the caller's security UTOKEN from the caller's environment and uses it when needed.

Table 20 lists the input callers must provide for each structure used for this request type and the output that will be provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields listed for each structure, see "Parameters" on page 152.

Table 20. Input and output for each structure used for the CeaTsoPing request type

Structure	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none">• eyecatcher• ceatso_version• ceatso_requesttype• ceatso_asid• ceatso_apptag• ceatso_stoken• ceatso_index	None
CeaTsoQueryStruct	Not used for this request type.	Not used for this request type.
CeaTsoErrorStruct	<ul style="list-style-type: none">• eyeCatcher• version	<ul style="list-style-type: none">• returnCode• reasonCode• diag

CeaTsoQuery - Querying the TSO/E address spaces

Use the CeaTsoQuery request type to obtain information from the CEA TSO/E address space manager about a TSO/E address space started by CEA.

To perform this request, the CEA TSO/E address space manager extracts the caller's security UTOKEN from the caller's environment and uses it when needed.

Table 21 on page 161 lists the input callers must provide for each structure used for this request type and the output that will be provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields listed for each structure, see "Parameters" on page 152.

Table 21. Input and output for each structure used for the CeaTsoQuery request type

Structure	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version • ceatso_requesttype • ceatso_asid • ceatso_apptag • ceatso_stoken • ceatso_index 	None
CeaTsoQueryStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version 	<p>If the return code is CEASUCCESS, the following fields are returned:</p> <ul style="list-style-type: none"> • ceatso_userid • ceatso_asid • ceatso_logonproc • ceatso_msgqueueid • ceatso_charset • ceatso_codepage • ceatso_screenrows • ceatso_screencols • ceatso_account • ceatso_group • ceatso_region • ceatso_apptag • ceatso_stoken • ceatso_index
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

CeaTsoQueryApp - Querying TSO/E sessions by application

Use the CeaTsoQueryApp request type to obtain information from the CEA TSO/E address space manager about all the TSO/E address spaces that CEA started that are associated with a specific application identifier.

To perform this request, the CEA TSO/E address space manager extracts the caller's security UTOKEN from the caller's environment and uses it when needed.

Table 22 on page 162 lists the input callers must provide for each structure used for this request type and the output that will be provided. No other fields in the structures are used. The value for the unused fields is indeterminate. For more details about the fields listed for each structure, see "Parameters" on page 152.

Attention: It is the caller's responsibility to free the storage associated with the query structures that are returned.

Table 22. Input and output for each structure used for the CeaTsoQueryApp request type

Structure	Required Input	Output
CeaTsoRequestStruct	<ul style="list-style-type: none"> • eyecatcher • ceatso_version • ceatso_requesttype • ceatso_asid • ceatso_numqueryreq • ceatso_apptag • ceatso_stoken • ceatso_index 	<p>If the return code is CEASUCCESS, the following field is returned:</p> <ul style="list-style-type: none"> • ceatso_numqueryrslt
CeaTsoQueryStruct	None	<p>If the return code is CEASUCCESS, an array of query structures are allocated and the following fields are returned for each:</p> <ul style="list-style-type: none"> • eyecatcher • ceatso_version • ceatso_userid • ceatso_asid • ceatso_logonproc • ceatso_msgqueueid • ceatso_charset • ceatso_codepage • ceatso_screenrows • ceatso_screencols • ceatso_account • ceatso_group • ceatso_region • ceatso_apptag • ceatso_stoken • ceatso_index
CeaTsoErrorStruct	<ul style="list-style-type: none"> • eyeCatcher • version 	<ul style="list-style-type: none"> • returnCode • reasonCode • diag

Return, reason, and diagnostic codes

When the CEATsoRequest API returns control to your program, the CEATsoErrorStruct structure contains the return, reason, and diagnostic codes that you can use to identify more information about any errors that occurred.

The codes the API returns are described in the following sections:

- “Return codes” on page 163
- “Reason codes” on page 163
- “Diagnostic codes” on page 167

Return codes

Table 23 lists and describes the return codes that are typically returned after the CEATsoRequest API processes a request.

Table 23. Return codes

Hexadecimal Return Code	Equate Symbol, Meaning, and Action
FFFFFFFF	<p>Equate symbol: CEAFailure</p> <p>Meaning: One or more errors occurred during CEATSOREQUEST processing.</p> <p>Action: Check the reason and diagnostic codes to obtain additional information, and correct any errors.</p>
00000000	<p>Equate symbol: CEASUCCESS</p> <p>Meaning: No errors occurred during CEATSOREQUEST processing. The meaning of a CEASUCCESS return code for each request type follows:</p> <ul style="list-style-type: none"> • CeaTsoStart. A new TSO/E address space was started, or the user was connected to a dormant TSO/E session. The caller can now read from and write to the z/OS UNIX message queue. • CeaTsoAttn. The attention interrupt request was sent to the specified TSO/E address space. • CeaTsoEnd. The specified TSO/E address space was ended or placed into a dormant state. If the session was ended, all associated resources were returned to the system. Otherwise, the resources were retained so that they can be reused when the user reconnects to the session. • CeaTsoPing. The ping request was performed, and the timestamp for the specified TSO/E session was updated. • CeaTsoQuery. The query completed with no errors. • CeaTsoQueryApp. The query by application completed with no errors. An array of query structures were allocated and populated with information about the sessions. <p>Action: None.</p>
00000004	<p>Equate symbol: CEAWARNING</p> <p>Meaning: One or more warnings occurred during CEATSOREQUEST processing.</p> <p>Action: Check the reason and diagnostic codes to obtain additional information, and correct any errors.</p>

Reason codes

Table 24 on page 164 lists and describes the reason codes that are typically returned after the CEATsoRequest API processes a request. Additional reason codes might also be returned from services that obtained an unexpected error. Those reason codes are not listed in the table.

Table 24. Reason codes

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
1000	<p>Equate symbol: CEATSOMSGQSERVICEFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: z/OS UNIX message queue processing failed.</p> <p>Action: Ensure that the CEA started task is TRUSTED. For more information about the RACF TRUSTED attribute, see the topic on associating started procedures and jobs with user IDs in <i>z/OS Security Server RACF System Programmer's Guide</i>, and the topic on using started procedures in <i>z/OS Security Server RACF Security Administrator's Guide</i>.</p>
1001	<p>Equate symbol: CEATSONOUSERIDFOUND</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: An input user ID value was expected, but not received.</p> <p>Action: Specify a user ID.</p>
1002	<p>Equate symbol: CEATSOMATCHMISSING</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: A user ID was expected, but not found in the session table.</p> <p>Action: Ensure that the user ID, STOKEN, and index specified are valid.</p>
1003	<p>Equate symbol: CEATSOSTOKENMISSING</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: An input STOKEN value was expected, but not received.</p> <p>Action: Specify a STOKEN.</p>
1004	<p>Equate symbol: CEATSOINDEXOUTOFRANGE</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: Input table index is too big or too small for the session table.</p> <p>Action: Specify a valid index. The index for the TSO/E address space should be between 1 and 2000.</p>
1005	<p>Equate symbol: CEATSOSStartFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: CEA could not create a TSO/E address space.</p> <p>Action: Ensure that sufficient system resources are available to create the TSO/E address space, and verify that the user is authorized to create address spaces.</p>
1006	<p>Equate symbol: CEATSOATTNFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: CEA could not issue a TSO/E attention interrupt.</p> <p>Action: Check the diagnostic codes to obtain additional information, and correct any errors.</p>

Table 24. Reason codes (continued)

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
1007	<p>Equate symbol: CEATSOENDFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: CEA could not end a TSO/E address space.</p> <p>Action: Check the diagnostic codes to obtain additional information, and correct any errors.</p>
1008	<p>Equate symbol: CEATSOQUERYFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: An attempt to query the session table failed.</p> <p>Action: Ensure that the input values you specified are valid. If the input values are valid, check the diagnostic codes to obtain additional information. Correct any errors.</p>
1009	<p>Equate symbol: CEATSOQUERYAPPFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: An attempt to query the session table for the TSO/E sessions that are associated with a specific application failed.</p> <p>Action: Ensure that the application identifier you specified is valid. If the application identifier is valid, check the diagnostic codes to obtain additional information. Correct any errors.</p>
100A	<p>Equate symbol: CEATSOPINGFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: Ping processing failed. Typically, this error occurs when the ping request is not issued from the security environment where the TSO/E address space was started or the user is not authorized to the application identified when the TSO/E address space was created.</p> <p>Note that the TSO/E address space is started or reconnected to using the security environment of the caller. If there is task-level security, it is used for the address space. Otherwise, the address space security environment is used. The user tokens (UTOKENs) from both environments are saved and are used to verify subsequent requests.</p> <p>Action: Issue the ping request from the security environment that was used when the TSO/E address space was started, and ensure that the user is authorized to the application specified when the address space was created.</p>
100B	<p>Equate symbol: CEATSOENDSENDLOGOFFFAILED</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The CANCEL command was issued to end the TSO/E address space because the LOGOFF command failed.</p> <p>Action: None.</p>

Table 24. Reason codes (continued)

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
100C	<p>Equate symbol: CEATSOBadAmode</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The call was invoked in the wrong AMODE. AMODE 64 is required.</p> <p>Action: Invoke the API in AMODE 64.</p>
100D	<p>Equate symbol: CEATSODisabled</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The dispatchable unit is not enabled.</p> <p>Action: Ensure that the dispatchable unit is enabled.</p>
100E	<p>Equate symbol: CEATSONotTaskMode</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The CEATsoRequest API was not invoked under task mode. The dispatchable unit mode must be task.</p> <p>Action: Ensure that the dispatchable unit is a task.</p>
100F	<p>Equate symbol: CEATSOFRSet</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The CEATsoRequest API was invoked under a functional recovery routine (FRR). No FRRs are allowed.</p> <p>Action: Ensure that no FRRs are invoked in your environment.</p>
1010	<p>Equate symbol: CEATSOLocked</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The caller is holding a system lock. No system locks are allowed.</p> <p>Action: Release the lock.</p>
1011	<p>Equate symbol: CEATSOXMMMode</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The CEATsoRequest API was invoked while running cross memory mode, which is not allowed. The API must be invoked in primary mode.</p> <p>Action: Invoke the API in primary mode.</p>
1013	<p>Equate symbol: CEATsoReqStructFieldBad</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: Input provided for a field in the CEATsoRequestStruct structure is not valid.</p> <p>Action: To identify the field that is not valid, see the diagnostic codes.</p>
1014	<p>Equate symbol: CEATsoBadQueryEyecatcher</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The eye catcher specified for the query structure is not valid. The expected value is CEAYTSOQ.</p> <p>Action: Specify CEAYTSOQ as the value for the eye catcher field.</p>

Table 24. Reason codes (continued)

Hexadecimal Reason Code	Equate Symbol, Meaning, and Action
1015	<p>Equate symbol: CEATsoBadQueryVersion</p> <p>Meaning: Error occurred during CEATSOREQUEST processing: The version specified for the query structure is not valid.</p> <p>Action: Specify a valid version number. The version numbers allowed are specified in the ceaytsor.h header file.</p>

Diagnostic codes

Table 25 lists and describes the diagnostic codes that are typically returned after the CEATsoRequest API processes a request. Additional diagnostic codes might also be returned from services that obtained an unexpected error. Those diagnostic codes are not listed in the table.

Table 25. Diagnostic code

Hexadecimal Diagnostic Code	Equate Symbol and Meaning
04	<p>Equate symbol: kCEATsoBadRacRouteExtr</p> <p>Meaning: The TSO/E address space was not started because an error occurred while trying to authenticate the caller. The CEA TSO/E address space service could not complete one of the following actions:</p> <ul style="list-style-type: none"> • Extract the security identity of the caller. • Log the caller into TSO/E. • Authorize the caller to a required resource. <p>The following fields are returned in the CEATsoErrorStruct structure:</p> <ul style="list-style-type: none"> • diag2 contains the SAF return code from RACRoute returned in R15. • diag3 contains the RACF or installation return code from the SAF parameter list. • diag4 contains the RACF or installation exit reason code from the SAF parameter list. <p>Note that a value is not always returned in diag2, diag3, and diag4.</p>
05	<p>Equate symbol: kCEATsoBadRacRouteCreate</p> <p>Meaning: An error was encountered when requesting verification of the newly created security identity.</p> <p>The following fields are returned in the CEATsoErrorStruct structure:</p> <ul style="list-style-type: none"> • diag2 contains the SAF return code from RACRoute returned in R15. • diag3 contains the RACF or installation return code from the SAF parameter list. • diag4 contains the RACF or installation exit reason code from the SAF parameter list.

Table 25. Diagnostic code (continued)

Hexadecimal Diagnostic Code	Equate Symbol and Meaning
0A	<p>Equate symbol: kCEATsoBadAddSession</p> <p>Meaning: Unable to create a new TSO/E address space.</p> <p>The return code received from the TSO/E session is provided in the diag2 field of the CEATsoErrorStruct structure. If the value in the diag2 field is 15, this indicates that the CEA TSO/E address space manager has created the maximum number of TSO/E sessions allowed. In that case, a message is also written to the z/OS console indicating that the session limit has been reached and user requests cannot be processed.</p>
0B	<p>Equate symbol: kCEATsoBadQuerySession</p> <p>Meaning: Unable to query the attributes of TSO/E sessions that are associated with a specific application.</p> <p>The return code received from the method is provided in the diag2 field of the CEATsoErrorStruct structure.</p>
0C	<p>Equate symbol: kCEATsoBadASCBS token</p> <p>Meaning: Unable to issue an attention interrupt or query the session table for information about the TSO/E address space because the STOKEN could not be found.</p>
0D	<p>Equate symbol: kCEATsoBadSessIndex</p> <p>Meaning: The value provided in the ceatso_index field in the CeaTsoRequestStruct is zero, which is not valid. The index must be greater than or equal to one.</p>
0F	<p>Equate symbol: kCEATsoBadLOGONMGCRCRE</p> <p>Meaning: The MGCRCRE service used to issue the start command to start a TSO/E address space failed.</p> <p>The register where MGCRCRE returned its return code is provided in the diag2 field of the CEATsoErrorStruct structure. In this case, the value in the diag2 field is R15 (register 15).</p>
10	<p>Equate symbol: SESS_SESSIONNOLONGERINTABLE</p> <p>Meaning: The TSO/E session no longer exists in the session table.</p>
11	<p>Equate symbol: kCEATsoBadSessENQreq</p> <p>Meaning: Unable to acquire the ENQ on the session table.</p> <p>The return code received from the method is provided in the diag2 field of the CEATsoErrorStruct structure.</p>
13	<p>Equate symbol: kCEATsoBadSessUpdateLastRef</p> <p>Meaning: The ping request failed because the CEA TSO/E address space manager was unable to update the last reference timestamp for that session.</p> <p>The return code received from the method is provided in the diag2 field of the CEATsoErrorStruct structure.</p>

Table 25. Diagnostic code (continued)

Hexadecimal Diagnostic Code	Equate Symbol and Meaning
14	<p>Equate symbol: kCEATsoBadQuerySessionForApptag</p> <p>Meaning: Unable to query the session table for the specified application identifier because an error occurred.</p> <p>The return code received from the method is provided in the diag2 field of the CEATsoErrorStruct structure.</p>
15	<p>Equate symbol: kCEATsoBadNumEntries</p> <p>Meaning: The number of entries found that match the query exceeds the maximum number of sessions that can be queried or exceeds the number of entries the query structure can accommodate.</p> <p>The number of entries found is provided in the diag2 field of the CEATsoErrorStruct structure.</p>
18	<p>Equate symbol: kCEATsoBadResmgrAdd</p> <p>Meaning: Unable to set the end of memory resource manager; an ABEND dump was taken.</p>
19	<p>Equate symbol: kCEATsoBadQueryAllSessions</p> <p>Meaning: Unable to perform a query of all TSO/E sessions in the session table. You must search for a specific TSO/E session, or search for TSO/E sessions that are associated with a specific application identifier.</p> <p>The return code received from the method is provided in the diag2 field of the CEATsoErrorStruct structure.</p>
1A	<p>Equate symbol: kCEATsoBadApptag</p> <p>Meaning: The value contained in the application identifier field is not valid.</p>
1B	<p>Equate symbol: kCEATsoBaduserid</p> <p>Meaning: The value contained in the user ID field is not valid.</p>
1C	<p>Equate symbol: kCEATsoBadlogonproc</p> <p>Meaning: The value contained in the logon procedure field is not valid.</p>
1F	<p>Equate symbol: kCEATsoBadscreenrows</p> <p>Meaning: The number of screen rows specified is out of range. The minimum number of screen rows is 24, and the maximum is 204.</p>
20	<p>Equate symbol: kCEATsoBadscreencols</p> <p>Meaning: The number of screen columns specified is out of range. The minimum number of screen columns is 80, and the maximum is 160.</p>
21	<p>Equate symbol: kCEATsoBadaccount</p> <p>Meaning: The value contained in the account field is not valid.</p>

Table 25. Diagnostic code (continued)

Hexadecimal Diagnostic Code	Equate Symbol and Meaning
22	Equate symbol: kCEATsoBadgroup Meaning: The value contained in the TSO/E group name field is not valid.
23	Equate symbol: kCEATsoBadregion Meaning: The value contained in the TSO/E region size field is not valid.
26	Equate symbol: kCEATsoBadCharsetCodepage Meaning: The value contained in the codepage field is not valid because no match was found in the Coded Character Set Identifiers (CCSID) table.
27	Equate symbol: kCEATsoBadregionsize Meaning: The value contained in the region size field is not valid because it exceeds the maximum allowable region size of 2,096,128.

CEAYTSOR header file

For the C programmer, include file ceaytsor.h defines the structures, functions, and macros used for the CEATsoRequest API. The header file is stored in partitioned data set SYS1.SIEAHDRV, and contains the following information.

```
#ifndef __ceaytsor__
#define __ceaytsor__

/***** START OF SPECIFICATIONS *****/
*
* DESCRIPTIVE NAME:  CEA TsoRequest structures
*
* ACRONYM:  CEAYTSOR
*
* STRUCT NAME:  None
*
* LABEL PREFIX:  None
*
* COMPONENT ID:  Common Event Adpater (CEA)
*
*****/

/****PROPRIETARY_STATEMENT*****/
/*
/*
/* LICENSED MATERIALS - PROPERTY OF IBM
/* COPYRIGHT IBM CORP. 2011, 2012
/*
/* STATUS= HBB7770
/*
/****END_OF_PROPRIETARY_STATEMENT*****/
/*
/*01* EXTERNAL CLASSIFICATION: PI
/*01* END OF EXTERNAL CLASSIFICATION:
/*
/*****/

/* $Id: ieacl1as2.ide, ieapr, osnp_v1r13.5 1.9 12/01/24 17:16:48 $ */
```

```

/*****
 * FUNCTION:
 *
 *
 * This header file defines the structures, functions
 * and macros used for CEATsoRequest() API.
 *
 * This support requires the setting of _XOPEN_SOURCE_EXTENDED
 *
 * RESTRICTIONS:
 *   None
 *
 * CHANGE-ACTIVITY:
 *
 *****/

/*****
          Constants
 *****/
#define CEATSOREQUEST_CURRENTVERSION 1
#define CEATSOQUERY_CURRENTVERSION 1
#define CEATSOERROR_CURRENTVERSION 1
#define CEATSODIAG_CURRENTVERSION 1
#define CEATSOREQUEST_EYECATCHER "CEAYTSOR"
#define CEATSOQUERY_EYECATCHER "CEAYTSOQ"
#define CEATSOERROR_EYECATCHER "CEAIERRO"

/*****
          CONSTANTS ceatso_requesttype;
          These are the request types used in the CEATsoRequest structure
 *****/
#define CeaTsoStart 1
#define CeaTsoEnd 2
#define CeaTsoQuery 3
#define CeaTsoAttn 4
#define CeaTsoPing 5
#define CeaTsoQueryApp 6

/*****
          CONSTANTS ceatso_flags
          These are the flag values used in the CEATsoRequest structure
 *****/
#define CEATSO_ABLOGOFF 0x8000 // Use Cancel to end the TSO session
#define CEATSO_NOREUSE 0x4000 // Do not reconnect an existing session

/*****
CEATsoRequestStruct_t

eyeCatcher - "CEAYTSOR"
version - CEATSOQUERY_CURRENTVERSION
request - request - uses CeaTso* constants

 *****/

struct CEATsoRequestStruct_s {
    char ceatso_eyecatcher[8]; /* eye catcher: CEAYTSOR */
    uint32_t ceatso_version; /* version number */
    uint32_t ceatso_requesttype; /* which type request */
    char ceatso_userid[8]; /* tso id */
    uint32_t ceatso_asid; /* tso asid */
    char ceatso_logonproc[8]; /* logon proc name */
    char ceatso_command[80]; /* unused */
    uint16_t ceatso_numqueryreq; /* caller num max query */
    uint16_t ceatso_numqueryrslt; /* actual num query */
    uint32_t ceatso_duration; /* unused */
    uint32_t ceatso_msgqueueid; /* msg queue id */

```

```

uint16_t      ceatso_charset;      /* callers character set */
uint16_t      ceatso_codepage;     /* callers code page */
uint16_t      ceatso_screenrows;   /* screen rows */
uint16_t      ceatso_screencols;   /* screen cols */
char          ceatso_account[40];  /* tso account number */
char          ceatso_group[8];     /* tso group name */
char          ceatso_region[7];    /* tso region size */
char          ceatso_instance[1];  /* tso instance number */
char          ceatso_apptag[8];    /* identity of caller */
char          ceatso_stoken[8];    /* tso asid stoken */
uint32_t      ceatso_ascbaddr;     /* tso ascb address */
uint16_t      ceatso_flags;        /* tso request flags */
uint16_t      ceatso_index;        /* tso session index */
char          rsvd1[8];            /* reserved space */
};
typedef struct CEATsoRequestStruct_s CEATsoRequestStruct_t;

```

```

/*****
CEATsoQueryStruct_t*

```

This structure is used to return Query results for the CEATsoRequest
CEATsoQuery

```

eyeCatcher - "CEAYTSOQ"
version    - 1

```

```

*****/
struct CEATsoQueryStruct_s{      /* query results */
char          ceatsoq_eyecatcher[8]; /* eye catcher: CEAYTSOQ */
uint32_t      ceatsoq_version;      /* version number */
uint32_t      ceatsoq_requesttype;  /* which type request */
char          ceatsoq_userid[8];    /* tso id */
uint32_t      ceatsoq_asid;         /* tso asid */
char          ceatsoq_logonproc[8]; /* logon proc name */
char          ceatsoq_command[80];  /* tso command */
uint16_t      ceatsoq_numqueryreq;  /* caller num max query */
uint16_t      ceatsoq_numqueryrs1t; /* actual num query */
uint32_t      ceatsoq_duration;     /* duration */
uint32_t      ceatsoq_msgqueueid;   /* msg queue id */
uint16_t      ceatsoq_charset;      /* callers character set */
uint16_t      ceatsoq_codepage;     /* callers code page */
uint16_t      ceatsoq_screenrows;   /* screen rows */
uint16_t      ceatsoq_screencols;   /* screen cols */
char          ceatsoq_account[40];  /* tso account number */
char          ceatsoq_group[8];     /* tso group name */
char          ceatsoq_region[7];    /* tso region size */
char          ceatsoq_instance[1];  /* tso instance number */
char          ceatsoq_apptag[8];    /* identity of caller */
char          ceatsoq_stoken[8];    /* tso asid stoken */
uint32_t      ceatsoq_ascbaddr;     /* tso ascb address */
uint16_t      ceatsoq_flags;        /* tso request flags */
uint16_t      ceatsoq_index;        /* tso session index */
char          rsvd1[8];            /*reserved space */
};
typedef struct CEATsoQueryStruct_s CEATsoQueryStruct_t;

```

```

/*****
CEATsoDiag_t

```

```

version      - version of CEADdiag_t
flags        - diagnostic flags
offset       - offset point to additional information
rsvd         - reserved for future use
diag1        - Used to hold return codes
diag2        - from system REXX scripts
diag3        - or other things outside of
diag4        - CEA control

```

```

    rsvd2          - reserved for future use
    messageArea   - Contains any output messages relating to error codes

* This structure is part of CEAEError, doesn't get its own eyecatcher
*****/

struct CEATsoDiag_s {
    uint8_t  version;
    uint8_t  flags1;
    uint16_t offset;
    uint8_t  diagid;
    char     rsvd[3];
    uint32_t diag1;
    uint32_t diag2;
    uint32_t diag3;
    uint32_t diag4;
    char     rsvd2[16];
    char     messageArea[256];
};
typedef struct CEATsoDiag_s CEATsoDiag_t;

/*****
CEAEError_t

eyeCatcher - "CEAIERR0"
version    - version of CEAEError_t
returnCode - function return code - duplicate of function return value
reasonCode - further explanation of a return code.
diag       - further explanation of a reason code.
*****/

struct CEATsoError_s {
    char     eyeCatcher[8];
    uint32_t version;
    int32_t  returnCode;
    uint32_t reasonCode;
    CEATsoDiag_t diag;
};
typedef struct CEATsoError_s CEATsoError_t;

/*****
Function prototype CEATsoRequest
*****/
#ifdef __cplusplus
extern "C" {
#endif
int32_t CEATsoRequest(CEATsoRequestStruct_t*,
                    CEATsoQueryStruct_t*,
                    CEATsoError_t*);
#ifdef __cplusplus
}
#endif

/*****
Diag Values

These are the possible values that can be returned in the Diag1
field in the CEAEError_t Diag structure returned from the
CEATsoRequest API

Note: Some duplication of codes exist but codes are unique per API
Request Type
*****/

#define kCEATsoBadRacRouteExtr    0X0004 //0004
#define kCEATsoBadRacRouteCreate 0X0005 //0005
#define kCEATsoBadAddSession     0X000A //0010

```

```

#define kCEATsoBadQuerySession      0X000B //0011
#define kCEATsoBadASCBStoken        0X000C //0012
#define kCEATsoBadSessIndex         0X000D //0013
#define kCEATsoBadRemoveSessEntry   0X000E //0014
#define kCEATsoBadLogonMGCRC        0X000F //0015
#define kCEATsoSessionNotFound      0X0010 //0016
#define kCEATsoBadSessENQreq        0X0011 //0017
#define kCEATsoBadSessDEQreq        0X0012 //0018
#define kCEATsoBadSessUpdateLR      0X0013 //0019
#define kCEATsoBadQuerySessApptag   0X0014 //0020
#define kCEATsoBadNumEntries        0X0015 //0021
#define kCEATsoBadMsgQDelete        0X0016 //0022
#define kCEATsoBadAppTag            0X0017 //0023
#define kCEATsoBadWiComCreate       0X0017 //0023
#define kCEATsoBadResmgrAdd         0X0018 //0024
#define kCEATsoBadQueryAllSessions  0X0019 //0025
#define kCEATsoBadApptag           0X001A //0026
#define kCEATsoBaduserid            0X001B //0027
#define kCEATsoBadlogonproc         0X001C //0028
#define kCEATsoBadcharset           0X001D //0029
#define kCEATsoBadcodepage          0X001E //0030
#define kCEATsoBadscreenrows        0X001F //0031
#define kCEATsoBadscreencols        0X0020 //0032
#define kCEATsoBadaccount           0X0021 //0033
#define kCEATsoBadgroup             0X0022 //0034
#define kCEATsoBadregion            0X0023 //0035
#define kCEATsoBadQueryEyecatcher   0X0024 //0036
#define kCEATsoBadQueryVersion      0X0025 //0037
#define kCEATsoBadCharsetCodepage   0X0026 //0038
#define kCEATsoBadregionsize        0X0027 //0039

#endif /* __ceaytsor__ */

```

CEAXRDEF header file

For the C programmer, include file ceaxrdef.h defines the return codes and reason codes that are associated with the CEA TSO/E address space manager services. The header file is stored in partitioned data set SYS1.SIEAHDRV, and contains the following information.

```

#ifndef __ceaxrdef__
#define __ceaxrdef__

/***** START OF SPECIFICATIONS *****/
*
* DESCRIPTIVE NAME:  CEA reason code definitions
*
* ACRONYM:  CEAXRDEF
*
* STRUCT NAME:  None
*
* LABEL PREFIX:  None
*
* COMPONENT ID:  Common Event Adpater (CEA)
*
*****/

/* $Id: ieaclas2.ide, ieapr, osnp_v1r13.5 1.9 12/01/24 17:16:48 $ */

/****PROPRIETARY_STATEMENT*****/
/*
/*
/* LICENSED MATERIALS - PROPERTY OF IBM
/* COPYRIGHT IBM CORP. 2011, 2012
*/

```

```

/*                                                                    */
/* STATUS= HBB7770                                                    */
/*                                                                    */
/****END_OF_PROPRIETARY_STATEMENT*****/
/*                                                                    */
/*01* EXTERNAL CLASSIFICATION: PI                                     */
/*01* END OF EXTERNAL CLASSIFICATION:                               */
/*                                                                    */
/*****/

/*****
* ceaxrdef.h header file
* -----
* This header file defines the reason codes associated with
* the Common Event Adapter (a.k.a. CEAS) client code.
*
*
* CHANGE-ACTIVITY:
*
*****/

// Completion Codes
#define CEASUCCESS      0
#define CEAFailure     -1
#define CEAWARNING      4

// Reason Codes
#define CEAUNAVAIL      0x100 //256
#define CEADUPPLICATENAME 0x101 //257
#define CEANOCONNAUTH  0x102 //258
#define CEANOACCESS    0x103 //259
#define CEABADPID      0x104 //260
#define CEABADHANDLE   0x105 //261
#define CEADUPESUB     0x106 //262
#define CEADUPHANDLER  0x107 //263
#define CEANOSUBSCRIBE 0x108 //264
#define CEANOMATCH     0x109 //265
#define CEASMLLBUFF    0x10A //266
#define CEANODATA      0x10B //267
#define CEADATATRUNC   0x10C //268 //returned on warning
#define CEAEVENTSMISSED 0x10D //269 //returned on warning
#define CEANOSUBAUTH   0x10E //270
#define CEABADPROTOCOL 0x10F //271
#define CEACOMMFAILURE 0x110 //272
#define CEASYSTEMFAILURE 0x111 //273
#define CEAINVALIDCLIENT 0x112 //274
#define CEASOFTWAREFAILURE 0x113 //275
#define CEABADHANDLEPTR 0x114 //276
#define CEASECURITYFAILURE 0x115 //277
#define CEAINVALIDCOMMAND 0x116 //278
#define CEAMAXCLIENTSCONNECTED 0x117 //279
#define CEANOTYETIMPLEMENTED 0x118 //280
#define CEABADREGVERSION 0x119 //281
#define CEAEFFFAILURE  0x11A //282
#define CEADYNEXFAILURE 0x11B //283
#define CEAEVENTSLOSTTRUNC 0x11C //284
#define CEAUSSSHUTDOWN 0x011D //285
#define CEANOENFEXITRTN 0x011E //286
#define CEASYSOPFORCEUNSUBSCRIBE 0x011F //287
#define CEASYSOPFORCEDISCONNECT 0x0120 //288
#define CEAFORCEMINMODE 0x0121 //289
#define CEAUSSNOTACTIVE 0x0122 //290
#define CEAMAXWTOSUBSCRIBED 0x0123 //291
#define CEAMAXEVENTSSUB 0x0124 //292
#define CEAMAXSUBCONNECTED 0x0125 //293
#define CEAMAXPGMSUBSCRIBED 0x0126 //294

```

```

#define CEANONAME 0x0200 //512
#define CEAINVALIDPARM 0x0201 //513
#define CEABADCONVERSION 0x0202 //514
#define CEANOTRECOGNIZED 0x0203 //515
#define CEANOTYPE 0x0204 //516
#define CEABADENFCODE 0x0205 //517
#define CEABADRETVERSION 0x0206 //518
#define CEABADEVENTVERSION 0x0207 //519
#define CEAINVALIDFORM 0x0208 //520
#define CEAINVALIDMODE 0x0209 //521
#define CEAHANDLERNOTFOUND 0x020A //522
#define CEAHANDLERNOTREENT 0x020B //523
#define CEAINVALIDHANDLER 0x020C //524
#define CEACONNECTNOTDEFSEC 0x020D //525
#define CEAEVENTNOTDEFSEC 0x020E //526
#define CEABADCLIENTNAME 0x020F //527
#define CEAINVALIDMSGID 0x0210 //528
#define CEABADADDRESS 0x0211 //529
#define CEAEVENTNOTALPHANUM 0x0212 //530
#define CEAEVENTHASBLANKS 0x0213 //531
#define CEAMAXTHRUPTREACHED 0x0214 //532
#define CEABADQMASK 0x0215 //533
#define CEABADBITCOMPARE 0x0216 //534
#define CEAMAXENFX 0x0217 //535
#define CEAREJECTENFX 0x0218 //536
#define CEATYPEENFXNOTSUPPORTED 0x0219 //537

#define CEAREQUESTNOTRECOGNIZED 0x0300 //768
#define CEAREQUESTNOTIMPLEMENTED 0x0301 //769
#define CEAPROPERTYSTRUCTBADPTR 0x0302 //770
#define CEAPROPERTYSTRUCTBADEYE 0x0303 //771
#define CEAPROPERTYSTRUCTBADVERSION 0x0304 //772
#define CEAPROPERTYBADRESOURCE 0x0305 //773
#define CEAPROPERTYNOMATCH 0x0306 //774
#define CEAPROPERTYSTRUCTEMPTY 0x0307 //775
#define CEAENVBAD 0x0308 //776
#define CEAFILTERSTRUCTBADEYE 0x0309 //777
#define CEAFILTERSTRUCTBADVERSION 0x030A //778
#define CEAFILTERBADRESOURCE 0x030B //779
#define CEAFILTERNOMATCH 0x030C //780
#define CEABADPARMPTR 0x030D //781
#define CEABADSSISUBSYSTEM 0x030E //782
#define CEABADSSICALL 0x030F //783
#define CEANOSSI 0x0310 //784
#define CEABADSSIENV 0x0311 //785
#define CEAENVBADSSI 0x0312 //786
#define CEANOFILTFORVERBOSE 0x0313 //787
#define CEAUNABLETOALLOCATE 0x0314 //788
#define CEANOTJOBSTERSEELEMENT 0x0315 //789
#define CEAJOBCHAINBROKEN 0x0316 //790
#define CEABADDATENV 0x0317 //791
#define CEASYSOUTCHAINBROKEN 0x0318 //792
#define CEANOTSYSOUTHRELEMENT 0x0319 //793
#define CEABADFREEPTR 0x031A //794
#define CEABADFREEBLK 0x031B //795
#define CEABADFREEENV 0x031C //796
#define CEAUNABLETOFREE 0x031D //797
#define CEABADIEFQR 0x031E //798
#define CEASSCHAINBROKEN 0x031F //799
#define CEAENVBADJSQY 0x0320 //800
#define CEABADFILTEROPER 0x0321 //801
#define CEABADS54SUBSYSTEM 0x0322 //802
#define CEABADS54CALL 0x0323 //803
#define CEANOS54 0x0324 //804
#define CEABADS54ENV 0x0325 //805
#define CEAENVBADS54 0x0326 //806
#define CEABADS54STOR 0x0327 //807

```

```

#define CEATIMEOUTMAXIMUMEXCEEDED 0x0328 //808
#define CEANEEDSYSOUTFILTER 0x0329 //809
#define CEABUFFERTOOLARGE 0x032A //810
#define CEACMDSDIAGRCSET 0x032B //811
#define CEACMDSAXREXXRCSET 0x032C //812
#define CEANOINSTRAUTH 0x032D //813
#define CEATOOMUCHDATA 0x032E //814
#define CEAFILTERNOTSUPPORTED 0x032F //815
#define CEAPRIMARYTYPEMISMATCH 0x0330 //816
#define CEABADSUBSYSTEM 0x0331 //817
#define CEUNABLETOALLOCATE2 0x0332 //818
#define CEABADBUFFER 0x0333 //819
#define CEATIMEOUTLESSTHANMINIMUM 0x0334 //820
#define CEACMDSNTAXERROR 0x0335 //821
#define CEACMDSHALTERROR 0x0336 //822
#define CEACMDSUNINITERROR 0x0337 //823
#define CEAFILTERBADCOMBO 0x0338 //824
#define CEACMDSTIMEDOUT 0x0339 //825
#define CEALLREQBLOCKSINUSE 0x033A //826
#define CEAIHQCLIENTABENDED 0x033B //827
#define CEAIHQARGSCANNOTACCESS 0x033C //828
#define CEALISTCANNOTACCESS 0x033D //829
#define CEAIHQSERVERABENDED 0x033E //830
#define CEANOTACTIVE 0x033F //831
#define CEABADIPQSERVERRC 0x0340 //832
#define CEAMEMORYALLOCATION 0x0341 //833
#define CEASDDIREMPTY 0x0342 //834
#define CEAAADFAILED 0x0343 //835
#define CEAINCIDENTSTRUCTBADEYE 0x0344 //836
#define CEAINCIDENTSTRUCTBADVERSION 0x0345 //837
#define CEAEERRORSTRUCTBADEYE 0x0346 //838
#define CEAEERRORSTRUCTBADVERSION 0x0347 //839
#define CEAINCINAMESTRUCTBADEYE 0x0348 //840
#define CEABADBRANCHFORIPCSSVR 0x0349 //841
#define CEABADENVFORMAR 0x034A //842
#define CEAOBJECTTYPEBADEYE 0x034B //843
#define CEAOBJECTTYPEBADVERSION 0x034C //844
#define CEAPROBNOTYPEBADEYE 0x034D //845
#define CEAPROBNOTYPEBADVERSION 0x034E //846
#define CEAMAXINSTANCENOSUPPORT 0x034F //847
#define CEAPDWKEYSTRUCTBADEYE 0x0350 //848
#define CEADIAGSTRUCTBADVERSION 0x0351 //849
#define CEADAEDSNNOTAVAILABLE 0x0352 //850
#define CEACANTFINDCOUNTRYCODE 0x0353 //851
#define CEACANTFINDBRANCHCODE 0x0354 //852
#define CEABADPARMLIST 0x0355 //853
#define CEABADPARM 0x0356 //854
#define CEAGENPREPAREDSDSNFAIL 0x0357 //855
#define CEAREXXENERROR 0x0358 //856
#define CEAXREXXERROR 0x0359 //857
#define CEAINTERNALBUFFEROVERRUN 0x035A //858
#define CEABADTIMEOUTPTR 0x035B //859
#define CEABADOUTPUTBUFFERPTR 0x035C //860
#define CEABADOUTPUTBUFFERLENPTR 0x035D //861
#define CEABADERRORPTR 0x035E //862
#define CEARECOVERYFAILURE 0x035F //863
#define CEABADACRO 0x0360 //864
#define CEABADVER 0x0361 //865
#define CEADMPINCIDENTNOTFOUND 0x0362 //866
#define CEAINVALIDINCIDENTKEY 0x0363 //867
#define CEABADERRO 0x0364 //868
#define CEASYSREXXNOTACTIVE 0x0365 //869
#define CEASYSREXXBADENVIRONMENT 0x0366 //870
#define CEAXEETIMEOUT 0x0367 //871
#define CEASYSREXXOVERLOADED 0x0368 //872
#define CEADATABADEYE 0x0369 //873
#define CEADATABADVERSION 0x036A //874

```

```

#define CEASYSDUMPBADEYE          0X036B //875
#define CEASYSDUMPBADVERSION      0X036C //876
#define CEAINCIDENTSTRUCTBADTYPE  0X036D //877
#define CEAMIGLIBNOTAPFAUTH       0X036E //878
#define CEANOSAFOPERLOGSNAP       0X036F //879
#define CEALOGGERNOTAVAIL         0X0370 //880
#define CEABADALLOCNEW            0X0371 //881
#define CEATERSEBADALLOC1         0X0372 //882
#define CEABADIXGCONN             0X0373 //883
#define CEABADIXGBRWSESTART       0X0374 //884
#define CEABADIXGBRWSEREAD        0X0375 //885
#define CEANOSNAPSHOT             0X0376 //886
#define CEAPDWBOBJECTNOTFOUND     0X0377 //887
#define CEAPDWBDIAGDATAEMPTY      0X0378 //888
#define CEAWRONGIBMPMRFORMAT      0X0379 //889
#define CEABADLEVELOFPREPARATION  0X037A //890
#define CEADAESYPTOMNOTVALID      0X037B //891
#define CEADAESYPTOMNOTFOUND      0X037C //892
#define CEAIPCSENQERROR           0X037D //893
#define CEASDDIROPENERROR         0X037E //894
#define CEAXMLINITFAILURE         0X037F //895
#define CEAXMLPARSEFAILURE        0X0380 //896
#define CEAXMLTERMFAILURE         0X0381 //897
#define CEAXMLTAGSTOODEEP         0X0382 //898
#define CEAXMLPARMSBADEYE        0X0383 //899
#define CEADATASPACEBADPTR        0X0384 //900
#define CEAPREPAREOBJINUSE        0X0385 //901
#define CEAPREPAREENQERR          0X0386 //902
#define CEACKSTBADREQ             0X0387 //903
#define CEACKSTBUFLLEN            0X0388 //904
#define CEACKSTIGGCSICALLABEND   0X0389 //905
#define CEACKSTBADCONTROLBLOCK    0X038A //906
#define CEACKSTINVALIDSIZETYPE    0X038B //907
#define CEACKSTINVALIDALLOCVALUE  0X038C //908
#define CEACKSTINVALIDIGGCSIENTRY 0X038D //909
#define CEACKSTIGGCSICALLFAIL    0X038E //910
#define CEACKSTUCBSCANFAIL       0X038F //911
#define CEACKSTUCBSCANABND       0X0390 //912

#define CEASETINCFSELBADEYE       0X0393 //915
#define CEASETINCFSELBADVERSION   0X0394 //916
#define CEASETINCFVALBADEYE       0X0395 //917
#define CEASETINCFVALBADVERSION   0X0396 //918
#define CEASETINCFVALDATATRUNC    0X0397 //919
#define CEAMIGRATEDDATASETS       0X0398 //920
#define CEAMIGRATEDDATASETSWHMERR 0X0399 //921

#define CEATSOMSGQSERVICEFAILED   0X1000 //4096
#define CEATSONOUSERIDFOUND       0X1001 //4097
#define CEATSOMATCHMISSING        0X1002 //4098
#define CEATSOSTOKENMISSING       0X1003 //4099
#define CEATSOINDEXOUTOFRANGE     0X1004 //4100
#define CEATSOStartFAILED         0X1005 //4101
#define CEATSOATTNFAILED          0X1006 //4102
#define CEATSOENDFAILED           0X1007 //4103
#define CEATSOQUERYFAILED         0X1008 //4104
#define CEATSOQUERYAPPFAILED      0X1009 //4105
#define CEATSOPIINGFAILED         0X100A //4106
#define CEATSOENDESENDLOGOFFFAILED 0X100B //4107
#define CEATSOBADAMODE            0X100C //4108
#define CEATSODISABLED            0X100D //4109
#define CEATSONOTTASKMODE         0X100E //4110
#define CEATSOFRRSET              0X100F //4111
#define CEATSOLOCKED              0X1010 //4112
#define CEATSOXMMODE              0X1011 //4113
#define CEATSOSESSTBLDSPFAILED    0X1012 //4114

```

```

#define CEATSOREQSTRUCTFIELDDBAD    0X1013 //4115
#define CEATSOBADQUERYEYECATCHER    0X1014 //4116
#define CEATSOBADQUERYVERSION       0X1015 //4117
#endif /* __ceaxrdef__ */

```

Programming example

The following example shows how to invoke the CEATsoRequest API from a C program. For a sample compile job that you can use to compile this sample program, see “Sample compile job” on page 196.

```

/*****/
/*
/* CEASAMPT.c   Sample code to demonstrate the
/*              CEATsoRequest() API for CEA HBB7780
/*              CEA TSO ADDRESS SPACE MANAGER
/*
/*
/*
/* Classification: Unclassified
/*
/*
/* Copyright: (C) Copyright IBM Corp. 2011, 2012
/*              Liscensed Materials - Property of IBM
/*
/*
/*
/* Change History:
/* $1.0  20110314  CYL:  Initial Version
/* $1.1  20111015  PDA2:  Sample Program
/*
/*****/

#define    _XOPEN_SOURCE
#define    _POSIXI_SOURCE          2

#define    SESS_SESSIONNOLONGERINTABLE  16
#define    SESS_MATCHMISSING            11
#define    SESS_INDEXOUTOFRANGE        13
#define    kMaximumSessions             50

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <env.h>
#include <iconv.h>
#include <sys/msg.h>
#include <sys/types.h>
#include <time.h>

#include "ceaytsor.h"
#include "ceaxrdef.h"

void  init_expected_values( void );
void  init_ceatso_struct( void );
void  print_request_struct( void );
void  print_query_struct( void );
void  print_error_struct( void );
int   send_message( void );
int   check_message( int, int );
int   verify_messages( int, int );
int   verify_attn_messages( int, int );
void  save_required_members( void );
void  init_required_members( void );

```

```

void set_required_members( void );

#define NUMVARS 56

struct message_queue_s {
    long int    message_type;
    char       message_text[200];
};
typedef struct message_queue_s    message_queue_t;

int           error_counter;        /* Total errors    */

CEATsoRequestStruct_t    ceatso_request;
CEATsoQueryStruct_t     ceatso_query;
CEATsoError_t           ceatso_error;

char          userid[8];
uint32_t     asid;
char         apptag[8];
uint32_t     ascbaddr;
int          index_value;        /* Save index value */
char        stoken[8];          /* Stoken buffer    */
char        *stoken_ptr;        /* Stoken pointer   */
char        *ptr;

message_queue_t    message_queue;
int               message_id;
size_t            message_size;
char             message_text[200];
int              wait_seconds;    /* Msg receive time */
int              sleep_time;

char            *tso_cmd_ptr;
char            tso_cmd[80] =
    "{\"TSO RESPONSE\":{\"VERSION\": \"0100\", \"DATA\": \"ALLOC DA\"}}";

int32_t         expected_rc;
uint32_t       expected_rsn;
uint32_t       expected_diag1;
uint32_t       expected_diag2;
uint32_t       expected_diag3;
uint32_t       expected_diag4;
uint32_t       reason_mask;
int            CeaTsoSamp1( void );

int main() {
    int rc;                /* Return code    */

    CeaTsoSamp1();        /* Invoke the sample code */

    return 0;
}

/*****
**
** Routine to initialize the expected return code,
** reason code and diag codes.
**
*****/
void init_expected_values( void ) {

    expected_rc    = CEASUCCESS;
    expected_rsn   = 0;
    expected_diag1 = 0;

```

```

    expected_diag2 = 0;
    expected_diag3 = 0;
    expected_diag4 = 0;

    return;
}

/*****
**
** Routine to initialize the CEA TSO request structure
** query structure and error structure for API call
**
**
*****/
void init_ceilso_struct( void ) {

    /* Initialize CEA TSO Request structure for CEATsoRequest() */
    memset(&ceilso_request, '\0', sizeof(CEATsoRequestStruct_t));

    strcpy(ceilso_request.ceilso_eyecatcher, CEATSOREQUEST_EYECATCHER);

    ceilso_request.ceilso_version = CEATSOREQUEST_CURRENTVERSION;

    ceilso_request.ceilso_requesttype = 0;

    /*
    ceilso_request.ceilso_asid = 0;
    */

    strcpy(ceilso_request.ceilso_userid, "IBMUSER ");
    strcpy(ceilso_request.ceilso_logonproc, "OMVS0803");
    memset(&ceilso_request.ceilso_command, ' ', 80);

    /*
    ceilso_request.ceilso_numqueryreq = 12;

    ceilso_request.ceilso_numqueryrslt = 12;

    ceilso_request.ceilso_duration = 0;

    ceilso_request.ceilso_msgqueueid = 0;
    */

    ceilso_request.ceilso_charset = 697;
    ceilso_request.ceilso_codepage = 1047;
    ceilso_request.ceilso_screenrows = 24;
    ceilso_request.ceilso_screencols = 80;
    memset(ceilso_request.ceilso_account, '\0', 40);
    memset(ceilso_request.ceilso_group, ' ', 8);
    strcpy(ceilso_request.ceilso_region, "2000000");

    /*
    memset(ceilso_request.ceilso_instance, ' ', 1);
    */

    strcpy(ceilso_request.ceilso_apptag, "IZUIS ");

    ceilso_request.ceilso_flags = CEATSO_ABLOGOFF;
}

```

```

/*
  memset(ceatso_request.ceatso_stoken, 0xFF, 8);

  ceatso_request.ceatso_ascbaddr = 0;

  ceatso_request.ceatso_index = 0;
*/

/* Initialize the CEA TSO Query structure for CEATsoRequest() */
memset(&ceatso_query, '\0', sizeof(CEATsoQueryStruct_t));

strcpy(ceatso_query.ceatsoq_eyecatcher, CEATSOQUERY_EYECATCHER);

memset(&ceatso_request.ceatso_command, ' ', 40);

/* Initialize the CEA TSO Error structure for CEATsoRequest() */
memset(&ceatso_error, 0x00, sizeof(CEATsoError_t));

strcpy(ceatso_error.eyeCatcher, CEAINCT_EYE_CEAIERRO);

ceatso_error.version = CEAIERRO_CURRENTVERSION;

return;
}

/*****
/**
/** Routine to print out the CEATsoRequest structure
/** used by CEATsoRequest( ) API.
/**
/**
*****/
void print_request_struct( void ) {
  int i;

  printf("\n\n\nCEATsoRequest structure\n\n");

  printf("sizeof(CEATsoRequestStruct_t) = %d\n\n",
        sizeof(CEATsoRequestStruct_t));

  printf("CeaTsoRequest Eyecatcher = ");

  ptr = ceatso_request.ceatso_eyecatcher;
  for ( i = 1; i <= 8; i++ )
    printf("%C", *ptr++);
  printf("\n");

  printf("CeaTsoRequest Version = %d\n",
        ceatso_request.ceatso_version);

  printf("CeaTsoRequest Requesttype = %d\n",
        ceatso_request.ceatso_requesttype);

  printf("CeaTsoRequest Userid = ");

  ptr = ceatso_request.ceatso_userid;
  for ( i = 1; i <= 8; i++ )
    printf("%C", *ptr++);
  printf("\n");

  printf("CeaTsoRequest Asid = %X\n",
        ceatso_request.ceatso_asid);

```

```

printf("CeaTsoRequest LogonProc      = ");

ptr = ceatso_request.ceatso_logonproc;
for ( i = 1; i <= 8; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoRequest Command        = ");

ptr = ceatso_request.ceatso_command;
for ( i = 1; i <= 40; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoRequest Numqueryreq     = %d\n",
       ceatso_request.ceatso_numqueryreq);

printf("CeaTsoRequest Numqueryrslt    = %d\n",
       ceatso_request.ceatso_numqueryrslt);

printf("CeaTsoRequest Duration        = %d\n",
       ceatso_request.ceatso_duration);

printf("CeaTsoRequest Msgqueueid      = %d\n",
       ceatso_request.ceatso_msgqueueid);

printf("CeaTsoRequest Charset          = %d\n",
       ceatso_request.ceatso_charset);

printf("CeaTsoRequest Codepage         = %d\n",
       ceatso_request.ceatso_codepage);

printf("CeaTsoRequest Screenrows       = %d\n",
       ceatso_request.ceatso_screenrows);

printf("CeaTsoRequest Screencols       = %d\n",
       ceatso_request.ceatso_screencols);

printf("CeaTsoRequest Account          = ");

ptr = ceatso_request.ceatso_account + 32;
for ( i = 1; i < 8; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoRequest Group            = ");

ptr = ceatso_request.ceatso_group;
for ( i = 1; i <= 8; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoRequest Region            = ");

ptr = ceatso_request.ceatso_region;
for ( i = 1; i <= 7; i++ )
    printf("%C", *ptr++);
printf("\n");

ptr = ceatso_request.ceatso_instance;
printf("CeaTsoRequest Instance        = %C\n", *ptr);

printf("CeaTsoRequest Apptag           = ");

ptr = ceatso_request.ceatso_apptag;
for ( i = 1; i <= 8; i++ )

```

```

    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoRequest Stoken          = ");

stoken_ptr = ceatso_request.ceatso_stoken;
for ( i = 1; i <= 8; i++)
    printf("%X ", *stoken_ptr++);
printf("\n");

printf("CeaTsoRequest ASCBaddr          = %8X\n",
       ceatso_request.ceatso_ascbaddr);

printf("CeaTsoRequest Flags              = %d\n",
       ceatso_request.ceatso_flags);

printf("CeaTsoRequest Index              = %d\n",
       ceatso_request.ceatso_index);

printf("\n");

return;
}

/*****
**
** Routine to print out the CEATsoQuery structure
** used by CEATsoRequest( ) API.
**
**
*****/
void print_query_struct( void ) {
    int i;

printf("\n\n\nCEATsoQuery structure\n\n");

printf("sizeof(CEATsoQueryStruct_t) = %d\n\n",
       sizeof(CEATsoQueryStruct_t));

printf("CeaTsoQuery Eyecatcher          = ");

ptr = ceatso_query.ceatsoq_eyecatcher;
for ( i = 1; i <= 8; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery Version              = %d\n",
       ceatso_query.ceatsoq_version);

printf("CeaTsoQuery Requesttype          = %d\n",
       ceatso_query.ceatsoq_requesttype);

printf("CeaTsoQuery Userid                = ");

ptr = ceatso_query.ceatsoq_userid;
for ( i = 1; i <= 8; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery Asid                  = %X\n",
       ceatso_query.ceatsoq_asid);

printf("CeaTsoQuery LogonProc              = ");

ptr = ceatso_query.ceatsoq_logonproc;
for ( i = 1; i <= 8; i++ )

```

```

    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery   Command           = ");

ptr = ceatso_query.ceatsoq_command;
for ( i = 1; i <= 40; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery   Numqueryreq       = %d\n",
       ceatso_query.ceatsoq_numqueryreq);

printf("CeaTsoQuery   Numqueryrslt     = %d\n",
       ceatso_query.ceatsoq_numqueryrslt);

printf("CeaTsoQuery   Duration          = %d\n",
       ceatso_query.ceatsoq_duration);

printf("CeaTsoQuery   Msgqueueid       = %d\n",
       ceatso_query.ceatsoq_msgqueueid);

printf("CeaTsoQuery   Charset           = %d\n",
       ceatso_query.ceatsoq_charset);

printf("CeaTsoQuery   Codepage         = %d\n",
       ceatso_query.ceatsoq_codepage);

printf("CeaTsoQuery   Screenrows       = %d\n",
       ceatso_query.ceatsoq_screenrows);

printf("CeaTsoQuery   Screencols       = %d\n",
       ceatso_query.ceatsoq_screencols);

printf("CeaTsoQuery   Account          = ");

ptr = ceatso_query.ceatsoq_account + 32;
for ( i = 1; i < 8; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery   Group             = ");

ptr = ceatso_query.ceatsoq_group;
for ( i = 1; i <= 8; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery   Region            = ");

ptr = ceatso_query.ceatsoq_region;
for ( i = 1; i <= 7; i++ )
    printf("%C", *ptr++);
printf("\n");

ptr = ceatso_query.ceatsoq_instance;
printf("CeaTsoQuery   Instance         = %C\n", *ptr);

printf("CeaTsoQuery   Apptag           = ");

ptr = ceatso_query.ceatsoq_apptag;
for ( i = 1; i <= 8; i++ )
    printf("%C", *ptr++);
printf("\n");

printf("CeaTsoQuery   Stoken             = ");

```

```

    stoken_ptr = ceatso_query.ceatsoq_stoken;
    for ( i = 1; i < 9; i++)
        printf("%X ", *stoken_ptr++);
    printf("\n");

    printf("CeaTsoQuery   ASCBaddr       = %8X\n",
           ceatso_query.ceatsoq_ascbaddr);

    printf("CeaTsoQuery   Flags         = %d\n",
           ceatso_query.ceatsoq_flags);

    printf("CeaTsoQuery   Index         = %d\n",
           ceatso_query.ceatsoq_index);

    printf("\n");

    return;
}

/*****
/**
/** Routine to print out the CEATsoError  structure      **/
/** used by CEATsoRequest( ) API.                      **/
/**                                                    **/
*****/
void print_error_struct( void ) {
    int    i;

    printf("\n\n\nCEATsoError  structure\n\n");

    printf("sizeof(CEATsoError_t)      = %d\n\n",
           sizeof(CEATsoError_t));

    printf("CEAError   Eyecatcher       = ");

    ptr = ceatso_error.eyeCatcher;
    for ( i = 1; i <= 8; i++)
        printf("%C", *ptr++);
    printf("\n");

    printf("CEAError   Version          = %8d\n",
           ceatso_error.version);

    printf("CEAError   ReturnCode(hex)   = %8X\n",
           ceatso_error.returnValue);

    printf("CEAError   ReasonCode(hex)   = %8X\n",
           ceatso_error.reasonCode);

    printf("CEAError   Diag.diag1(hex)   = %8X\n",
           ceatso_error.diag.diag1);

    printf("CEAError   Diag.diag2(hex)   = %8X\n",
           ceatso_error.diag.diag2);

    printf("CEAError   Diag.diag3(hex)   = %8X\n",
           ceatso_error.diag.diag3);

    printf("CEAError   Diag.diag4(hex)   = %8X\n",
           ceatso_error.diag.diag4);

    printf("\n");

    return;
}

```

```

}

/*****
/**                                     **/
/** Verify messages                       **/
/**                                     **/
*****/
int verify_messages(int message_id, int wait_seconds ) {
    int rc;
    char *string1;
    char *string2;
    char *string3;
    char *string4;
    char *string5;
    char *string6;

    if ( ceatso_request.ceatso_requesttype == CeaTsoStart ) {
        rc = check_message(message_id, wait_seconds);
        string1 = "LOGON IN PROGRESS";
        if ( rc != 0 || strstr(message_text, string1) == NULL ) {
            printf(" Failed to receive %s message.\n\n", string1);
            return 99;
        }

        rc = check_message(message_id, wait_seconds);
        string2 = "NO BROADCAST MESSAGES";
        if ( rc != 0 || strstr(message_text, string2) == NULL ) {
            printf(" Failed to receive %s.\n\n", string2);
            return 99;
        }

        rc = check_message(message_id, wait_seconds);
        string3 = "READY ";
        if ( rc != 0 || strstr(message_text, string3) == NULL ) {
            printf(" Failed to receive %s prompt.\n\n", string3);
            return 99;
        }

        rc = check_message(message_id, wait_seconds);
        string4 = "HIDDEN";
        string5 = "FALSE";
        if ( rc != 0
            strstr(message_text, string4) == NULL ||
            strstr(message_text, string5) == NULL ) {
            printf(" Failed to receive %s : %s message.\n\n",
                string4, string5 );
            return 99;
        }
    }

    if ( ceatso_request.ceatso_requesttype == CeaTsoAttn ) {
        rc = check_message( message_id, wait_seconds );
        string6 = "ENTER DATA SET NAME OR * -";
        if ( rc != 0
            strstr(message_text, string6) == NULL ) {
            printf(" Failed to receive %s message.\n\n", string6);
            return 99;
        }

        rc = check_message(message_id, wait_seconds);
        string4 = "HIDDEN";
        string5 = "FALSE";
        if ( rc != 0
            strstr(message_text, string4) == NULL ||
            strstr(message_text, string5) == NULL ) {
            printf(" Failed to receive %s : %s message.\n\n",
                string4, string5 );
            return 99;
        }
    }
}

```

```

        string4, string5 );
    return 99;
}
}

return 0;
}

/*****
/**                                     **/
/** Verify messages after Attn         **/
/**                                     **/
*****/
int verify_attn_messages(int message_id, int wait_seconds ) {
    int rc;
    char *string1;
    char *string2;
    char *string3;

    rc = check_message(message_id, wait_seconds);
    string1 = "READY ";
    if ( rc != 0 || strstr(message_text, string1) == NULL ) {
        printf("    Failed to receive %s prompt after Attn.\n\n",
            string1);
        return 99;
    }

    rc = check_message(message_id, wait_seconds);
    string2 = "HIDDEN";
    string3 = "FALSE";
    if ( rc != 0
        strstr(message_text, string2) == NULL ||
        strstr(message_text, string3) == NULL ) {
        printf("    Failed to receive %s : %s message.\n\n",
            string2, string3);
        return 99;
    }

    return 0;
}

/*****
/**                                     **/
/** Check message text                 **/
/**                                     **/
*****/
int check_message( int message_id, int wait_seconds ) {
    int rc;
    size_t iconv_rc;
    ssize_t msg_rc;
    iconv_t cd;
    char *input_ptr;
    char *output_ptr;
    size_t input_msgsize;
    size_t output_msgsize;
    time_t wait_time;
    time_t start_time;
    time_t receive_time;

    message_size = sizeof(message_queue_t) - sizeof(long int);

```

```

memset(&message_text, '\0', message_size);

time(&start_time);

/* -6 should include 2 and 3 */
message_queue.message_type = (long int)-6;

sleep_time = 2;
msg_rc = 0;

/* Must include IPC_NOWAIT flag, otherwise could hang */
/* the program execution when no msg sending back. */
do {
    msg_rc = msgrcv(message_id, &message_queue, message_size,
                    message_queue.message_type, MSG_NOERROR | IPC_NOWAIT);
    sleep( sleep_time);
    wait_time = time(&receive_time) - start_time;
} while ( wait_time <= wait_seconds && msg_rc <= 0 );

if ( msg_rc == -1 ) {
    printf("\n\nReceive message failed with\n");
    printf("    msg_rc = %d ", msg_rc);
    printf("    Wait time = %d seconds\n", wait_time);
    printf("    Errno = %X", errno);
    printf("    Errno_Jr = %X\n\n", __errno2());
    return 99;
}
else
    printf("    Received Message in %d seconds.\n",
           wait_time);

if ( (rc = setenv("_ICONV_UCS2", "D", 1)) != 0 ) {
    printf("\n    setenv( ) failed with ");
    printf("    rc = %d ", rc);
    printf("    Errno = %X ", errno);
    printf("    Errno_Jr = %X\n\n", __errno2());
    return rc;
}

if ( (cd = iconv_open("IBM-1047", "UTF-8")) == (iconv_t)-1 ) {
    printf("    iconv_open( ) failed with ");
    printf("    Errno = %X ", errno);
    printf("    Errno_Jr = %X\n\n", __errno2());
    return 99;
}

input_ptr = message_queue.message_text;
output_ptr = message_text;

input_msgsize = msg_rc;
output_msgsize = msg_rc;

if ((iconv_rc = iconv(cd, &input_ptr, &input_msgsize, &output_ptr,
                     &output_msgsize)) == (size_t)-1 ) {
    printf("    iconv( ) failed with ");
    printf("    rc = %d ", iconv_rc);
    printf("    Errno = %X ", errno);
    printf("    Errno_Jr = %X\n\n", __errno2());
    return 99;
}

if ( (rc = iconv_close( cd )) == -1 ) {
    printf("    iconv_close( ) failed with ");
    printf("    rc = %d ", rc);
    printf("    Errno = %X ", errno);
    printf("    Errno_Jr = %X\n\n", __errno2());
}

```

```

    return rc;
}

printf("    Received Message Type:    %2d\n",
       message_queue.message_type);
printf("    Received Message Length:  %d\n", strlen(message_text));
printf("    Received Message Text:  \n");
printf("        %s\n", message_text);
printf("\n");

return 0;
}

/*****
**
** Send TSO command and check the proper message received
**
**
*****/
int send_message( void ) {
    int rc;
    size_t iconv_rc;
    iconv_t cd;
    size_t input_msgsize;
    size_t output_msgsize;
    char *input_ptr;
    char *output_ptr;

    message_size = sizeof(message_queue_t) - sizeof(long int);
    memset(&message_queue.message_text, '\0', message_size);
    memset(&message_text, '\0', message_size);

    strcpy(message_text, tso_cmd);

    if ( (cd = iconv_open("UTF-8", "IBM-1047")) == (iconv_t)-1 ) {
        printf("    iconv_open( ) failed with    ");
        printf("    Errno = %X    ", errno);
        printf("    Errno_Jr = %X\n\n", __errno2());
        return 99;
    }

    input_ptr = message_text;
    output_ptr = message_queue.message_text;

    input_msgsize = strlen(message_text);
    output_msgsize = input_msgsize;

    if ((iconv_rc = iconv(cd, &input_ptr, &input_msgsize, &output_ptr,
        &output_msgsize)) == (size_t)-1 ) {
        printf("    iconv( ) failed with    ");
        printf("    rc = %d    ", iconv_rc);
        printf("    Errno = %X    ", errno);
        printf("    Errno_Jr = %X\n\n", __errno2());
        return 99;
    }

    if ( (rc = iconv_close( cd )) == -1 ) {
        printf("    iconv_close( ) failed with    ");
        printf("    rc = %d    ", rc);
        printf("    Errno = %X    ", errno);
        printf("    Errno_Jr = %X\n\n", __errno2());
        return rc;
    }
}

```

```

message_queue.message_type = (long int)7;
message_size = strlen(message_queue.message_text);

rc = msgsnd(message_id, &message_queue, message_size, 0);

return rc;
}

/*****
/**
/** Save some required members of request structure
/** for ATTN and END process
/**
/**
*****/
void save_required_members( void ) {
    int i;

/* Not required input for End
if ( ceatso_request.ceatso_requesttype == CeaTsoEnd ) {
    strcpy(userid, ceatso_request.ceatso_userid);
    strcpy(apptag, ceatso_request.ceatso_apptag);
}

if ( ceatso_request.ceatso_requesttype == CeaTsoAttn )
    asid = ceatso_request.ceatso_asid;
*/

    asid = ceatso_request.ceatso_asid;

    stoken_ptr = stoken;
    ptr = ceatso_request.ceatso_stoken;
    for ( i = 1; i < 9; i++)
        *stoken_ptr++ = *ptr++;

    ascbaddr = ceatso_request.ceatso_ascbaddr;

    index_value = ceatso_request.ceatso_index;

/*
printf("\nSave the following value:\n");
*/

/* Not required input for End
if ( ceatso_request.ceatso_requesttype == CeaTsoEnd ) {
    printf("    userid      = ");

    ptr = userid;
    for ( i = 1; i <= 8; i++ )
        printf("%C", *ptr++);
    printf("\n");

    printf("    apptag      = ");

    ptr = apptag;
    for ( i = 1; i <= 8; i++ )
        printf("%C", *ptr++);
    printf("\n");
}
*/

/*
printf("    asid        = %X\n", asid);

ptr = ceatso_request.ceatso_stoken;
printf("    stoken      = ");

```

```

    for ( i = 1; i < 9; i++)
        printf("%X ", *ptr++);
    printf("\n");

    printf("    ascdaddr    =   %X\n", ascbaddr);

    printf("    index_value =   %X\n", index_value);

    printf("\n");
*/

    return;

}

/*****
**
** Initialize some required members of request structure
** for ATTN and END process
**
**
*****/
void init_required_members( void )    {
    int    i;

    memset(ceatso_request.ceatso_eyecatcher, 'F', 8);

    ceatso_request.ceatso_version = 0;

    if ( ceatso_request.ceatso_requesttype == CeaTsoAttn )
        ceatso_request.ceatso_asid = 0;

/*
    if ( ceatso_request.ceatso_requesttype == CeaTsoEnd ) {
        memset(ceatso_request.ceatso_userid, 'F', 8);
        memset(ceatso_request.ceatso_apptag, 'F', 8);
    }
*/

    memset(ceatso_request.ceatso_stoken, 0xFF, 8);

    ceatso_request.ceatso_ascbaddr = 0;

    ceatso_request.ceatso_index = 0;

    /* Initialize the CEA TSO Error structure for CEATsoRequest() */
    memset(&ceatso_error, 0x00, sizeof(CEATsoError_t));

    return;

}

/*****
**
** Set some required members of request structure back
** to the original value for ATTN and END process
**
**
*****/
void set_required_members( void )    {
    int    i;

    strcpy(ceatso_request.ceatso_eyecatcher, CEATSOREQUEST_EYECATCHER);

    ceatso_request.ceatso_version = CEATSOREQUEST_CURRENTVERSION;

```

```

/*
  if ( ceatso_request.ceatso_requesttype == CeaTsoEnd ) {
    strcpy(ceatso_request.ceatso_userid, userid);
    strcpy(ceatso_request.ceatso_apptag, apptag);
  }
*/

  if ( ceatso_request.ceatso_requesttype == CeaTsoAttn )
    ceatso_request.ceatso_asid = asid;

  stoken_ptr = stoken;
  ptr = ceatso_request.ceatso_stoken;
  for ( i = 1; i < 9; i++)
    *ptr++ = *stoken_ptr++;

  ceatso_request.ceatso_ascbaddr = ascbaddr;

  ceatso_request.ceatso_index = index_value;

  /* Initialize the CEA TSO Error structure for CEATsoRequest() */
  memset(&ceatso_error, 0x00, sizeof(CEATsoError_t));
  strcpy(ceatso_error.eyeCatcher, CEAINCT_EYE_CEAIERRO);
  ceatso_error.version = CEAIERRO_CURRENTVERSION;

  return;
}

/*****
**
** CeaTsoSamp1: Sample code to invoke CEATsoRequest() to start
** a CEA TSO Session send it an Attn interrupt the end the TSO
** session.
**
** Results are returned in the error structure
**
*****/
int CeaTsoSamp1( ) {
  int      i;
  int      rc;

  printf("=====\n");
  printf("==      Start CeaTsoRequest( ) Example      ==\n");
  printf("=====\n");
  printf("\n");

  printf("CEATSORequest( ) Start session.\n\n");
  init_ceatso_struct( );
  init_expected_values( );
  ceatso_request.ceatso_requesttype = CeaTsoStart;

  CEATsoRequest(&ceatso_request, &ceatso_query, &ceatso_error);

  if ( ceatso_error.returnCode == expected_rc    &&
      ceatso_error.reasonCode == expected_rsn    &&
      ceatso_error.diag.diag1 == expected_diag1  &&
      ceatso_error.diag.diag2 == expected_diag2  &&
      ceatso_error.diag.diag3 == expected_diag3  &&
      ceatso_error.diag.diag4 == expected_diag4  )
    printf(" Verifying logon messages.\n\n");
  else {
    error_counter = error_counter + 1;

```

```

printf("CEATsoRequest( ) Start session failed.\n\n\n");
print_error_struct( );
print_request_struct( );
printf("\nVariation %d failed.\n\n\n", variation_id);
printf("\n\n");
return error_counter;
}

wait_seconds = 8;
message_id = ceatso_request.ceatso_msgqueueid;
rc = verify_messages( message_id, wait_seconds );

if ( rc == 0 )
printf("\nCEATsoRequest( ) Start seesion successful.\n\n");
else {
error_counter = error_counter + 1;
printf("CEATsoRequest( ) Start failed to receive the message ");
printf("with rc = %d.\n\n\n", rc);
printf("\nVariation %d failed.\n\n\n", variation_id);
printf("\n\n");
return error_counter;
}

save_required_members( );

ceatso_request.ceatso_requesttype = CeaTsoAttn;

rc = send_message( );

if ( rc == 0 ) {
printf("\n\nSend TSO Command Successful.\n\n");
printf(" Send Message Type: %2d\n",
message_queue.message_type);
printf(" Send Message Length: %d\n",
strlen(message_queue.message_text));
printf("\n");
}
else {
printf("\nSend message failed with ");
printf(" rc = %d ", rc);
printf(" Errno = %X ", errno);
printf(" Errno_Jr = %X\n\n", __errno2());
error_counter = error_counter + 1;
printf("\nVariation %d failed.\n\n\n", variation_id);
printf("\n\n");
return error_counter;
}

rc = verify_messages(message_id, wait_seconds);

if ( rc == 0 )
printf("\n\nCEATsoRequest( ) Attn starts.\n\n");
else {
error_counter = error_counter + 1;
printf("\nVariation %d failed.\n\n\n", variation_id);
printf("\n\n");
return error_counter;
}

ceatso_request.ceatso_requesttype = CeaTsoAttn;
set_required_members( );
init_expected_values( );
strcpy(ceatso_request.ceatso_eyecatcher, CEATSOREQUEST_EYECATCHER);

```

```

CEATsoRequest(&ceatso_request, &ceatso_query, &ceatso_error);

if ( ceatso_error.returnCode == expected_rc    &&
     ceatso_error.reasonCode == expected_rsn  &&
     ceatso_error.diag.diag1 == expected_diag1 &&
     ceatso_error.diag.diag2 == expected_diag2 &&
     ceatso_error.diag.diag3 == expected_diag3 &&
     ceatso_error.diag.diag4 == expected_diag4 )
    printf(" Verifying messages after Attn.\n\n");
else {
    error_counter = error_counter + 1;
    printf("CEATsoRequest( ) Attn failed.\n\n");
    print_error_struct( );
    print_request_struct( );
    printf("\nVariation %d failed.\n\n\n", variation_id);
    return error_counter;
}

rc = verify_attn_messages(message_id, wait_seconds);

if ( rc == 0 )
    printf("\nCEATsoRequest( ) Attn successful.\n\n");
else {
    error_counter = error_counter + 1;
    printf("CEATsoRequest( ) Attn failed.\n\n");
    print_error_struct( );
    print_request_struct( );
    printf("\nVariation %d failed.\n\n\n", variation_id);
    return error_counter;
}

printf("\n\nCEATsoRequest( ) End    starts.\n");
set_required_members( );
init_expected_values( );
ceatso_request.ceatso_requesttype = CeaTsoEnd;

CEATsoRequest(&ceatso_request, &ceatso_query, &ceatso_error);

if ( ceatso_error.returnCode == expected_rc    &&
     ceatso_error.reasonCode == expected_rsn  &&
     ceatso_error.diag.diag1 == expected_diag1 &&
     ceatso_error.diag.diag2 == expected_diag2 &&
     ceatso_error.diag.diag3 == expected_diag3 &&
     ceatso_error.diag.diag4 == expected_diag4 )
    printf("\n\n\nCEATsoRequest( ) End    session successful.\n");
else {
    error_counter = error_counter + 1;
    printf("\n\nCEATsoRequest( ) End session failed.\n\n");
    print_request_struct( );
    print_error_struct( );
    printf("\nVariation %d failed.\n\n\n", variation_id);
    return error_counter;
}

if ( ceatso_error.returnCode == CEASUCCESS )
    printf("\n\n\nVariation %d succeeded.\n\n\n\n", variation_id);
else {
    error_counter = error_counter + 1;
    printf("\n\n\nVariation %d failed.\n\n\n\n", variation_id);
}

printf("=====\n");

```

```

printf("==      Finished Start  CeaTsoRequest( ) Example  \n");
printf("===== \n");
printf("\n\n\n\n");

return  error_counter;

}

```

Sample compile job

For C programmers, you can use the following sample compile job to compile the sample program. For more details about the sample program, see "Programming example" on page 179.

```

/* rexx */
/* c89/cc/c++ */
/* dbx needs -g or -Wc,debug */
/* list\(.\/) */
/* export _C89_STEPS='-1'    enable all steps, inc prelinker */
/* export _C89_TMPS ='-3'    prelinker will write composite .p file*/

'c89 -oceanamt -v -g -Wc,LP64,SHOW,SO,AGGR,XREF,NOOFF,NOOPT,EXP,LIST\(.\/
SSCOMM,DLL,STA,'LANGLVL(EXTENDED)',WARN64
      -Wl,LP64,map,xref
      ceasamt.c ceasapit.x

'
'ls -gatlrE ceasamt.* ceasamt'

```

Part 6. zEnterprise Data Compression (zEDC)

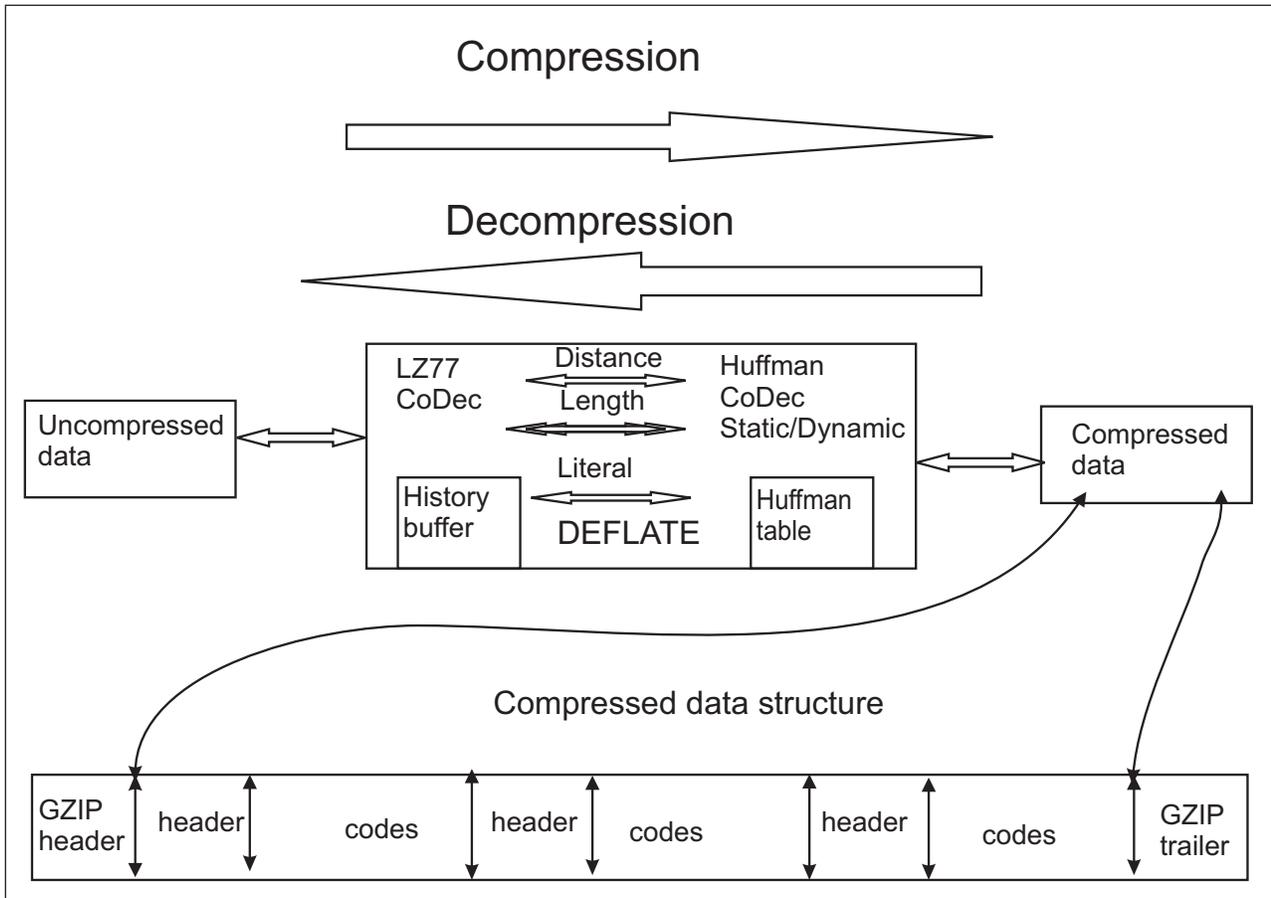
Chapter 13. Overview and planning of zEnterprise Data Compression (zEDC)

In today's z/OS environment, many installations want to compress certain types of data to occupy less space while its not in use, and then restore the data when necessary. Using zEnterprise Data Compression (zEDC) to compress data might help to reduce CPU cost and elapsed time of data compression compared to traditional software-based compression services, such as CSRCEserv and CSRcMPsc. zEDC can also lower the cost of applications using host-based compression that are currently running on z/OS.

zEDC supports the DEFLATE compression data format, which compresses data using the following algorithms, defined by RFC 1951:

- LZ77
 - Replaces repeated string with length, back pointer pairs.
 - Points back up to 32K.
- Huffman coding
 - Variable length encoding of characters.
 - Minimize bit length of stream of characters by assigning shorter codes to frequent characters.
 - Data and length, back pointer pairs are Huffman encoded.

For more details, check IETF standard RFC 1951 at <http://www.ietf.org/rfc/rfc1951.txt>.



Requirements for zEnterprise Data Compression

zEDC requires the following:

- z/OS V2R1 (or later) operating system.
- IBM zEnterprise EC12 CPC (with GA2 level microcode) or zBC12 CPC, or later.
- zEDC Express feature. This System z compression accelerator can improve the speed of data compression and is sharable across up to 15 partitions and up to 8 cards per CPC.
- zEDC Express software feature enabled in an IFAPRDxx parmlib member.
- Adequate 64-bit real storage configured to this z/OS image.

Planning for zEnterprise Data Compression

zEDC is established by launching either an unauthorized or authorized interface:

- Unauthorized interface for zEDC:
 - zlib for zEDC:
 - zlib is an OpenSource data compression library supporting the DEFLATE compressed data format.
 - The zlib compression library provides in-memory compression and decompression functions, including integrity checks of the uncompressed data. For additional information about zlib, see <http://zlib.net/>.
- System z authorized interfaces for zEDC:

- Requires supervisor state and supports task and SRB mode.
- Allows application buffers to be directly read by and written to by compression accelerator hardware, allowing the application to avoid a data move, but also adding complexity to managing I/O buffers.
- Operates on independent requests:
 - A deflate request produces a full DEFLATE block.
 - An inflate request consumes a full DELFATE block.
- Provides software inflate capability to maintain data access when System z compression accelerator hardware is not available.
- Additional method with the option to use zEDC:
 - SMF compression. Use the COMPRESS and PERMFIX keywords in the SMFPRMxx parmlib member to compress data before writing to a log stream. For additional information, see *z/OS MVS System Management Facilities (SMF)* and *z/OS MVS Initialization and Tuning Reference*.

Table 26. Comparison table between unauthorized and System z authorized interfaces for zEDC

Options	Unauthorized interfaces for zEDC	System z authorized interfaces for zEDC
Language	C	Any language that can call OS callable services
Data streaming	zlib-style data streams supported. Data can be broken up across requests as needed, but has to be within the minimum input buffer limit.	Each request is independent and handled as a single DEFLATE block. Inflate requests must receive single complete DEFLATE block.
Buffer management	Data move to device driver managed buffer.	Application buffer directly used by System z hardware.
Co-existence support	Both inflate and deflate are completed in software when hardware is not available.	Inflate completed in software when hardware is not available.
Authorization	Controlled by SAF-protected FACILITY class resource FPZ.ACCELERATOR.COMPRESSION.	Supervisor state.

Chapter 14. Application interfaces for zEnterprise Data Compression

This topic describes the following interfaces, considerations, and samples for zEnterprise Data Compression (zEDC):

- Invoking unauthorized interface for zEDC:
 - “zlib for zEnterprise Data Compression”
- Invoking System z authorized interfaces for zEDC:
 - “System z authorized compression services” on page 208
 - “FPZ4RZV — Rendezvous compression service” on page 208
 - “FPZ4PRB — Probe device availability compression service” on page 211
 - “FPZ4RMR — Memory registration compression service” on page 212
 - “FPZ4DMR — Deregister memory compression service” on page 214
 - “FPZ4ABC — Submit compression request” on page 215
 - “FPZ4URZ — Unrendezvous compression request” on page 220

Invoking unauthorized interfaces for zEnterprise Data Compression

zlib for zEnterprise Data Compression

The zlib data compression library provides in-memory compression and decompression functions, including integrity checks of the uncompressed data. A modified version of the zlib compression library is used by zEDC. The IBM-provided zlib compatible C library provides a set of wrapper functions that use zEDC compression when appropriate and when zEDC is not appropriate, software-based compression services are used.

The zlib wrapper functions use the following criteria to determine if zEDC can be used for compression:

- The system requirements for zEDC have been met. See “Requirements for zEnterprise Data Compression” on page 200 for the details.
- For a deflate stream, the parameters specified on `deflateInit2()` are supported by zEDC. For an inflate stream, all the parameters specified on `inflateInit2()` are supported. See “Standard zlib functions” on page 204 for the details.
- Because there are overhead costs when communicating with the hardware, on the first call to deflate or inflate a data stream, the provided input is checked to ensure that it is sufficiently large enough to make it worthwhile to use zEDC. If the data stream is large enough, zEDC is used. If the data stream is small, it might cost more to compress the data stream with zEDC so software-based compression services are used. **Note:** This check is only performed on the first call to deflate or inflate a data stream.

If any of these criteria are not met, the zlib wrapper function calls the standard zlib functions to process the data stream in software.

Once zEDC is used as the compression mechanism (for example, after the first call to inflate or deflate the data stream is completed), you cannot change the compression method to software-based compression services. At the same time, if software-based compression services are used as the compression mechanism (for

example, after the first call to inflate or deflate the data stream is completed), you cannot change the compression method to zEDC.

Note: Once a data stream starts using zEDC for compression, if a function is called that cannot be supported by zEDC or the zEDC hardware becomes unavailable, the unsupported function returns an error return code.

Standard zlib functions

The following table contains the standard zlib functions and whether they are supported using zEDC:

Table 27. Standard zlib functions and whether they are supported using zEDC

zlib function	zEDC-supported	Details
zlibVersion	Supported.	Returns '1.2.7-zEDC'
deflateInit	Supported.	
deflate	All flush modes are supported.	If the input buffer size is smaller than the minimum threshold for zEDC on the first call to deflate (compress) a data stream, the data stream is compressed using traditional software-based compression.
deflateEnd	Supported.	
inflateInit	Supported.	
inflate	Supported if the flush mode is one of the following: <ul style="list-style-type: none"> • z_no_flush • z_sync_flush • z_finish 	If either the input buffer size is smaller than a minimum threshold for zEDC or the flush mode is z_block or z_trees on the first call to inflate (decompress) a data stream, the data stream is decompressed using traditional software-based decompression. On subsequent calls to inflate a data stream, if the flush mode is z_block or z_trees and the stream is using zEDC decompression, Z_STREAM_ERROR is returned
inflateEnd	Supported.	
deflateInit2	Support is based on the input parameters.	<p>Input parameters:</p> <p>level This option is ignored for zEDC and does not affect the software or zEDC compression decision. This option is supported for zlib software compression.</p> <p>method Must be Z_DEFLATED.</p> <p>windowBits Must be -15 for raw deflate, 15 for zlib header and trailer, or 31 for gzip header and trailer. For all other windowBits values, the data stream uses traditional software-based compression.</p> <p>memLevel This option is ignored for zEDC and does not affect the software or zEDC compression decision. This option is supported for zlib software compression.</p> <p>strategy Use Z_DEFAULT_STRATEGY or Z_FIXED for zEDC. All other options use traditional software-based compression.</p>
deflateSetDictionary	Supported.	This option is supported for zEDC when called before the first deflate call for the data stream and is not supported after the first call to deflate.
deflateCopy	Supported.	

Table 27. Standard zlib functions and whether they are supported using zEDC (continued)

zlib function	zEDC-supported	Details
deflateReset	Supported.	
deflateParams	Support is based on the input parameters.	Input parameters: Level This option is ignored for zEDC. Strategy Use Z_DEFAULT_STRATEGY or Z_FIXED for zEDC. All other options use traditional software-based compression.
deflateTune	Supported.	This option only applies to traditional software-based compression. zEDC accepts the call, but none of the parameters apply to zEDC.
deflateBound	Supported.	
deflatePending	Supported.	
deflatePrime	Not supported for zEDC.	Returns Z_STREAM_ERROR if the stream is using zEDC.
deflateSetHeader	Supported.	
inflateInit2	Supported.	
inflateSetDictionary	Supported if called immediately after a call to inflate the data stream that returns Z_NEED_DICT.	Otherwise, Z_STREAM_ERROR is returned if the data stream is attempting to use zEDC decompression.
InflateSync	Supported.	
inflateCopy	Supported.	
inflateReset	Supported.	
inflatateReset2	Supported.	
inflatePrime	Not supported for zEDC.	Returns Z_STREAM_ERROR if the stream is using zEDC decompression.
inflateMark	Not supported for zEDC.	Returns Z_STREAM_ERROR if the stream is using zEDC decompression.
inflateGetHeader	Supported.	
inflateBackInit	Not supported for zEDC.	InflateBackInit forces stream to software-based compression.
inflateBack	Not supported for zEDC.	
zlibCompileFlags	Supported.	
compress	Supported.	
compress2	Supported.	Level is ignored if using zEDC.
compressBound	Supported.	
uncompress	Supported.	
gz* routines	Not supported for zEDC.	Uses software-based compression for inflate and deflate functions.
checksum functions	Not supported for zEDC.	Checksum functions calculate the checksum values using software-based compression services.

IBM-provided zlib compatible C library

The IBM-provided zlib compatible C library provides the following query functions in addition to the standard zlib functions:

deflateHwAvail(*buflen*)

Determines if the compression accelerator is available for a deflate operation. The input parameter *buflen* is an integer that represents the input buffer size of the first deflate request. The function returns an integer with a value of 1 if the compression accelerator will be used for the deflate operation or a value of 0 if software will be used instead.

inflateHwAvail(*buflen*)

Determines if the compression accelerator is available for an inflate operation. The input parameter *buflen* is an integer that represents the input buffer size of the first inflate request. The function returns an integer with a value of 1 if the compression accelerator will be used for this inflate operation or a value of 0 if software will be used instead.

hwCheck(*strm*)

Determines if a zlib stream is using the compression accelerator or software compression. The input parameter *strm* is a pointer to a zlib `z_stream` structure to check. The function returns an integer with a value of 0 if the stream has gone to the compression accelerator, a value of 1 if the stream is pending to go to the compression accelerator, but still could fall back to software compression, a value of 2 if the stream has gone to software compression, or `Z_STREAM_ERROR` if the stream has not been initialized correctly.

Running zlib

To compress data with zEDC, your installation must meet the system requirements. See “Requirements for zEnterprise Data Compression” on page 200 for the system requirements for zEDC.

To use the IBM-provided zlib compatible C library for data compression or data expansion services, follow these steps:

1. Link or re-link applications to use the IBM-provided zlib.

The IBM-provided zlib is an archive file in the z/OS UNIX System Services file system and can be statically linked into your applications. The paths for the zlib archive file and the zlib header files are:

Path for the zlib archive file:

`/usr/lpp/hzc/lib/libzz.a`

Path for the zlib header files:

`/usr/lpp/hzc/include/`

Note: When a new IBM service is provided for zlib, all applications that statically link zlib must re-link in order to use the updated IBM-provided zlib and take advantage of the new function.

2. Provide System Authorization Facility (SAF) Access:
 - Access to zEDC Express is protected by the SAF FACILITY resource class: `FPZ.ACCELERATOR.COMPRESSION`.
 - Give READ access to `FPZ.ACCELERATOR.COMPRESSION` to the identity of the address space that the zlib task will run in.

Note: The access check is performed during the first call in a given task. The results of that first check are cached for the duration of the task.

3. Use the z/OS UNIX environmental variable, `_HZC_COMPRESSION_METHOD`, to control if zEDC is used for data compression.

Note: If the value of *software* is set, software-based compression services are used. All other values result in the default behavior of attempting to use zEDC for data compression.

4. Ensure that adequately sized input buffers are available. If the input buffer size falls below the minimum threshold, data compression occurs using zlib software compression and not zEDC. This threshold can be controlled at a system level using the PARMLIB member IQPPRMxx.
5. Allocate the correct amount of storage for I/O buffers. The zEDC requests generated by zlib use predefined I/O buffer pools. The size of these I/O buffer pools can be set using PARMLIB member IQPPRMxx.

When zlib is statically linked into an application that runs on software or hardware that is not compatible with zEDC, zlib uses the following compression and decompression:

Table 28. Compression and decompression with zlib

Hardware level	z/OS level	zEDC Express	Description
zEC12 (with GA2 level microcode)	z/OS V2R1	Active	zEDC is used for both data compression and decompression.
zEC12 (with GA2 level microcode)	z/OS V2R1	Not Active	Requirements are not met for zEDC. When zEDC Express is not available, traditional software zlib is used for compression and decompression.
Pre-zEC12 (with GA2 level microcode)	z/OS V2R1 or pre-z/OS V2R1	N/A	Requirements are not met for zEDC. When zEDC Express is not available, traditional software zlib is used for compression and decompression.

zEDC error handling:

- If a System z compression accelerator is unavailable, data compression requests transfer to another System z compression accelerator configured to the same partition. These request transfers are transparent to the application.
- If all System z compression accelerators are unavailable, an error message is sent to the application.

Invoking System z authorized interfaces for zEnterprise Data Compression

This topic describes how to invoke System z authorized interfaces for zEnterprise Data Compression by:

- “System z authorized compression services” on page 208
 - “FPZ4RZV — Rendezvous compression service” on page 208
 - “FPZ4PRB — Probe device availability compression service” on page 211
 - “FPZ4RMR — Memory registration compression service” on page 212
 - “FPZ4DMR — Deregister memory compression service” on page 214
 - “FPZ4ABC — Submit compression request” on page 215
 - “FPZ4URZ — Unrendezvous compression request” on page 220

To compress data with zEDC, your installation must meet the system requirements. See “Requirements for zEnterprise Data Compression” on page 200 for the system requirements for zEDC.

All z/OS exploitation of zEDC handles mixed hardware and software levels. Compatibility APAR OA41245 provides software decompression for installations running with z/OS V1R13 or V1R12. The same software decompression is also provided for installations running z/OS V2R1 on pre-IBM zEnterprise EC12 (with GA2 level microcode). This allows access to compressed data on all combinations of environments.

Table 29. Compression and decompression with System z authorized interfaces for zEDC

Hardware level	z/OS level	zEDC Express	Description
zEC12 (with GA2 level microcode)	z/OS V2R1	Active	zEDC is used for both data compression and decompression.
zEC12 (with GA2 level microcode)	z/OS V2R1	Not Active	Requirements are not met for zEDC. Software-based decompression services for zEDC Express compressed data are used because zEDC Express compression is not available.
Pre-zEC12 (with GA2 level microcode)	z/OS V2R1	N/A	Requirements are not met for zEDC. Software-based decompression services for zEDC Express compressed data are used because zEDC Express compression is not available.
Pre-zEC12 (with GA2 level microcode)	Pre-z/OS V2R1	N/A	Requirements are not met for zEDC. Software-based decompression services for zEDC Express compressed data are used because zEDC Express compression is not available. Note: APAR OA41245 is required to use the software-based decompression services.

System z authorized compression services

The following compression services are available when using System z authorized interfaces for zEDC:

- “FPZ4RZV — Rendezvous compression service”
- “FPZ4PRB — Probe device availability compression service” on page 211
- “FPZ4RMR — Memory registration compression service” on page 212
- “FPZ4DMR — Deregister memory compression service” on page 214
- “FPZ4ABC — Submit compression request” on page 215
- “FPZ4URZ — Unrendezvous compression request” on page 220

FPZ4RZV — Rendezvous compression service

Description: The FPZ4RZV service performs the required setup and initialization of the compression services for an exploiter. The scope is the address space of the application and it is valid for the life of the Cross Memory Resource Owner (CMRO) task.

Notes:

1. A maximum of 255 rendezvous tokens are supported per each address space. This allows multiple applications to exploit the compression driver so each can maintain their own rendezvous scope.
2. All 64-bit storage is obtained with the MEMLIMIT=NO option.

Table 30. Environment for the FPZ4RZV service

Environmental factor	Requirement
Minimum authorization:	Supervisor State with Key 0
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	64-bit
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Table 31. Parameters for the FPZ4RZV service

Name	Type	Input/Output	Description
<i>ApplicationId</i>	Fixed(32)	Input	The application type to use. 0x01 is the application type for zEDC.
FPZ4RZV_options	Bit(64)	Input	Options for the FPZ4RZV service: SoftwareInflate (X'80000000 00000000') Allows compression requests to fall back to software inflation when no compression devices are available. EnableABCScatter (X'40000000 00000000') Allows compression requests to use the FPZ4ABC compression service to submit work with scatter/gather lists. FailOnNoDevices (X'20000000 00000000') If specified, compression requests fail when no compression devices are available. If FailOnNoDevices is not specified, a valid rendezvous token is returned even if no compression devices are currently available. This returned rendezvous token is used for all other services. PlusOne (X'08000000 00000000') If specified, compression requests will only use zEDC Express Adapters with the March 31, 2014 Firmware MCL release, or later. RmrEntriesExact (X'04000000 00000000') If specified, the <i>rmr_entries</i> parameter represents the maximum number of outstanding memory registrations for this rendezvous. If this limit is exceeded, the FPZ4RMR service may fail the request.
<i>userid</i>	Char(8)	Input	An eight character EBCDIC string identifying the user.
<i>rmr_entries</i>	Fixed(32)	Input	The estimated number of FPZ4RMR compression service calls to be performed that helps to size the tables used until the maximum number of registrations is reached. This is an optional parameter. The value of the <i>rmr_entries</i> parameter can be anywhere between 1 and 64K. The default is 128. Define <i>rmr_entries</i> as integer data of length 32.
Rendezvous token	Char(16)	Output	This is the token that must be passed to all FPZ services.

FPZ4RZV

Table 31. Parameters for the FPZ4RZV service (continued)

Name	Type	Input/ Output	Description
Return code	Fixed(31)	Output	The return code for the service.
Reason code	Fixed(32)	Output	The reason code for the service.

Table 32. Return and reason codes for the FPZ4RZV service

Hexadecimal return code	Reason code	Meaning and action
00	0000	Meaning: The call completed successfully. Action: None.
04	0000	Meaning: No zEDC devices are available. zEDC support is active so it is possible that zEDC devices might become available in the future. Action: If zEDC devices are available to this system, perform diagnostics to determine the reason for the failure.
04	0102	Meaning: No zEDC devices are available because the system requirements for zEDC were not met. See "Requirements for zEnterprise Data Compression" on page 200 for the details. A 'thin' rendezvous was created. Action: None.
08	0000	Meaning: No zEDC devices are available because the system requirements for zEDC were not met. This is the result of RvzFailOnNoDev being ON or SoftwareInflate being OFF when on downlevel hardware or software. See "Requirements for zEnterprise Data Compression" on page 200 for the details. No rendezvous token is returned. Action: None.
0C	0201	Meaning: Invalid parameter combination. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0207	Meaning: The calling environment is invalid. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0210	Meaning: <i>rmr_entries</i> specified an invalid value. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0226	Meaning: Invalid application specified. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
10	0301	Meaning: An internal error caused recovery to be entered. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Table 32. Return and reason codes for the FPZ4RZV service (continued)

Hexadecimal return code	Reason code	Meaning and action
10	0303	<p>Meaning: The maximum number of rendezvous tokens have been reached for the address space.</p> <p>Action: Determine if the calling program is at fault because of a coding error. If there is no coding error, another program might be consuming all the rendezvous tokens for the address space. Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

FPZ4PRB — Probe device availability compression service

Description: The FPZ4PRB service checks for the required hardware and software needed for zEDC. This service returns successful if they are available to the system. See “Requirements for zEnterprise Data Compression” on page 200 for the system requirements for zEDC.

Table 33. Environment for the FPZ4PRB service

Environmental factor	Requirement
Minimum authorization:	Supervisor State with Key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	64-bit
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Table 34. Parameters for the FPZ4PRB service

Name	Type	Input/Output	Description
<i>ApplicationId</i>	Fixed(32)	Input	The application type to use. 0x01 is the application type for zEDC.
FPZ4PRB_options	Bit(64)	Input	Options for the FPZ4PRB service: PlusOne (X'80000000 00000000') If specified, only zEDC Express Adapters with the March 31, 2014 Firmware MCL release, or later, will be honored. The value returned in <i>NumDevices</i> will only indicate this subset of devices.
<i>NumDevices</i>	Fixed(32)	Output	The number of devices available for this application.
Return code	Fixed(31)	Output	The return code for the service.
Reason code	Fixed(32)	Output	The reason code for the service.

Table 35. Return and Reason Codes for the FPZ4PRB service

Hexadecimal Return Code	Reason Code	Meaning and Action
00	0000	<p>Meaning: Devices are available.</p> <p>Action: None.</p>

FPZ4PRB

Table 35. Return and Reason Codes for the FPZ4PRB service (continued)

Hexadecimal Return Code	Reason Code	Meaning and Action
08	0900	Meaning: The z/OS software level is not correct for zEDC. See “Requirements for zEnterprise Data Compression” on page 200 for the details. Action: None.
08	0901	Meaning: The hardware level is not correct for zEDC. See “Requirements for zEnterprise Data Compression” on page 200 for the details. Action: None.
08	0902	Meaning: No zEDC devices are available. The hardware is at the correct level, but no zEDC devices were available. Action: If zEDC devices are available to this system, perform diagnostics to determine the reason for the failure.
08	0903	Meaning: zEDC devices were available during this IPL at some point, but there are no zEDC devices available now. Action: Perform diagnostics to determine the reason for the failure.

FPZ4RMR — Memory registration compression service

Description: The FPZ4RMR service registers a segment of memory for use by zEDC Express. The result is that this storage becomes fixed. The data area passed to FPZ4RMR must be page-aligned, and the size must be a multiple of a page boundary.

Note: This is not compatible with existing page fix services. This storage is eligible to be used for I/O as a result of this service.

Table 36. Environment for the FPZ4RMR service

Environmental factor	Requirement
Minimum authorization:	Supervisor State with Key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	64-bit
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Table 37. Parameters for the FPZ4RMR service

Name	Type	Input/Output	Description
<i>ApplicationId</i>	Fixed(32)	Input	The application type to use. 0x01 is the application type for zEDC.
FPZ4RMR_options	Bit(64)	Input	There are no supported options for the FPZ4RMR service.
Rendezvous token	Char(16)	Input	The rendezvous token.
Data@	Ptr(64)	Input	The address of the data area to register.

Table 37. Parameters for the FPZ4RMR service (continued)

Name	Type	Input/ Output	Description
DataLen	Fixed(64)	Input	The length of the data area to register.
Reserved	Fixed(32)	Input	Reserved. Must be 0.
DataKey	Fixed(8)	Input	The key of the data area to register. The format of this parameter is $0xk0$, where k represents the key of the data area.
RMR Token	Char(8)	Output	The region memory registration token associated with this data area. This token needs to be passed to the FPZ4ABC service when this data area is used as input or output.
Return code	Fixed(31)	Output	The return code for the service.
Reason code	Fixed(32)	Output	The reason code for the service.

Table 38. Return and Reason Codes for the FPZ4RMR service

Hexadecimal Return Code	Reason Code	Meaning and Action
00	0000	Meaning: The call completed successfully. Action: None.
08	0000	Meaning: Memory can not be registered because of lack of hardware support. Action: None.
08	0900	Meaning: Incorrect software level for zEnterprise data compression accelerator support. Action: None.
0C	0207	Meaning: The calling environment is invalid. Action: Determine if the calling program is at fault because of a coding error.
0C	0208	Meaning: An invalid rendezvous token was passed. Action: Check that the application successfully called the FPZ4RZV service.
0C	021D	Meaning: The supplied region was not CONTROL(AUTH). Action: Determine if the calling program is at fault because of a coding error.
0C	021E	Meaning: The supplied region address is incorrect. It might not have been page-aligned. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.
0C	021F	Meaning: The region length is invalid. It is possible that it is not a multiple of page size. Action: Determine if the calling program is at fault because of a coding error.

FPZ4RMR

Table 38. Return and Reason Codes for the FPZ4RMR service (continued)

Hexadecimal Return Code	Reason Code	Meaning and Action
0C	0220	Meaning: There is a region key mismatch. Action: Determine if the calling program is at fault because of a coding error.
0C	0226	Meaning: An invalid application ID was encountered. Action: Determine if the calling program is at fault because of a coding error.
0C	0227	Meaning: Rendezvous was not created with data space support. Action: Determine if the calling program is at fault because of a coding error.
10	0301	Meaning: An internal error has occurred. Action: Determine if the calling program is at fault because of a coding error.
10	0304	Meaning: Compression services were not initialized. Rendezvous was not called. Action: Check that the application successfully called the FPZ4RZV service.
10	0305	Meaning: Capacity has been reached for memory registrations. Action: Determine if the calling program is at fault because of a coding error.
10	0306	Meaning: There is not enough DMA memory available. Action: Determine if the calling program is at fault because of a coding error.

FPZ4DMR — Deregister memory compression service

Description: The FPZ4DMR service unregisters a segment of memory for use by zEDC Express. The result is that this storage becomes unfixed.

Table 39. Environment for the FPZ4DMR service

Environmental factor	Requirement
Minimum authorization:	Supervisor State
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	64-bit
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Table 40. Parameters for the FPZ4DMR service

Name	Type	Input/Output	Description
<i>ApplicationId</i>	Fixed(32)	Input	The application type to use. 0x01 is the application type for zEDC.

Table 40. Parameters for the FPZ4DMR service (continued)

Name	Type	Input/ Output	Description
FPZ4DMR_options	Bit(64)	Input	There are no supported options for the FPZ4DMR service.
Rendezvous token	Char(16)	Input	The rendezvous token.
RMR token	Char(8)	Input	The region memory registration (RMR) token associated with this data area to be unregistered.
Return code	Fixed(31)	Output	The return code for the service.
Reason code	Fixed(32)	Output	The reason code for the service.

Table 41. Return and Reason Codes for the FPZ4DMR service

Hexadecimal Return Code	Reason Code	Meaning and Action
00	0000	Meaning: The call completed successfully. Action: None.
08	0900	Meaning: Incorrect software level for zEnterprise data compression accelerator support. Action: None.
0C	0207	Meaning: The calling environment is invalid. Action: Determine if the calling program is at fault because of a coding error.
0C	0208	Meaning: An invalid rendezvous token was passed. Action: Check that the application successfully called the FPZ4RZV service.
0C	0209	Meaning: An invalid RMR token was provided. Action: Determine if the calling program is at fault because of a coding error.
10	0301	Meaning: An internal error has caused recovery to be entered. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.
10	0304	Meaning: Compression services were not initialized. Rendezvous was not called. Action: Check that the application successfully called the FPZ4RZV service.

FPZ4ABC — Submit compression request

Description: The FPZ4ABC service submits a single autonomous compression request for one or more DEFLATE blocks. The input and output buffers can be either direct buffers or scatter/gather lists.

Table 42. Environment for the FPZ4ABC service

Environmental factor	Requirement
Minimum authorization:	Supervisor State with Key 0

FPZ4ABC

Table 42. Environment for the FPZ4ABC service (continued)

Environmental factor	Requirement
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	64-bit
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Table 43. Parameters for the FPZ4ABC service

Name	Type	Input/Output	Description
<i>ApplicationId</i>	Fixed(32)	Input	The application type to use. 0x01 is the application type for zEDC.
FPZ4ABC_options	Bit(64)	Input	Options for the FPZ4ABC service: Inflate (X'80000000 00000000') When ON, specifies that this is an inflation request. Input Scatter List (X'40000000 00000000') When ON, the area pointed to by input@ is a scatter/gather list. Output Scatter List (X'20000000 00000000') When ON, the area pointed to by output@ is a scatter/gather list.
Rendezvous token	Char(16)	Input	The rendezvous token.
Input@	Ptr(64)	Input	The address of the input area or input scatter/gather list.
Output@	Ptr(64)	Input	The address of the output area or output scatter/gather list.
Input@RMR Token	Char(8)	Input	The region memory registration (RMR) token for the input area or area pointed to by the input scatter/gather list.
Output@RMR Token	Char(8)	Input	The region memory registration (RMR) token for the output area or area pointed to by the output scatter/gather list.
InputLen	Fixed(64)	Input	The length of the area pointed to by Input@. In the event that a scatter/gather list was provided using Input@, the total length of the areas provided by the scatter/gather areas must be provided.
OutputLen	Fixed(64)	Input	The length of the area pointed to by Output@. In the event that a scatter/gather list was provided using Output@, the total length of the areas provided by the scatter/gather areas must be provided.
GeneratedOutputLen	Fixed(64)	Output	This length describes how much output was generated and stored in either the Output@ or the scatter/gather list specified by Output@. This length spans across scatter/gather entries.
Return code	Fixed(31)	Output	The return code for the service.
Reason code	Fixed(32)	Output	The reason code for the service.

The FPZ4ABC service allows for the input and output areas to span several non-contiguous areas. The header of the FPZ4ABC list is immediately followed by the list entries. **Note:** All entries in the scatter/gather list must be associated with the same RMR token.

Scatter/gather lists have alignment rules and every entry in the scatter/gather list is checked for the following conditions:

- The start of the first buffer in the list can be on any byte boundary.
- The end of the first buffer must be on the required byte boundary.
- The start / end of the intermediate buffers must be on the required byte boundary.
- The start of the last buffer must be on the required byte boundary.
- The end of the last buffer can be on any boundary.

All required boundaries are on 128-byte alignment. A maximum of 8 scatter/gather entries are allowed.

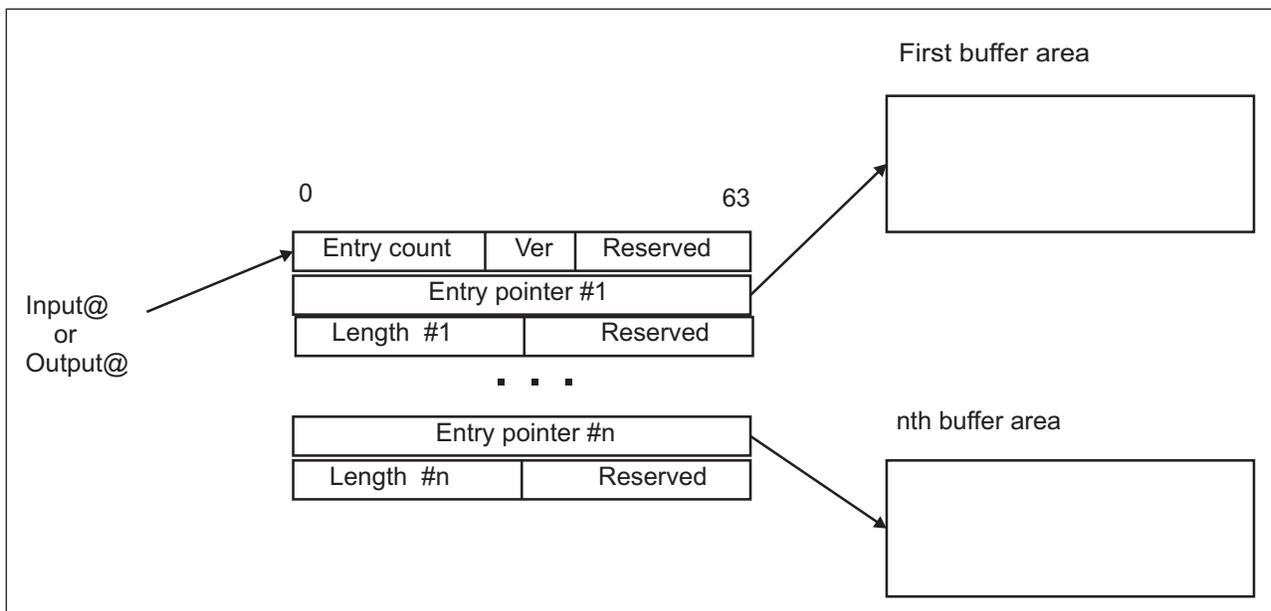


Table 44. Header elements in the FPZ4ABC-generated list

Name	Type	Description
# Of Entries	Fixed(32)	The number of entries in the list.
Version	Fixed(8)	The version associated with the list.
Reserved	Char(3)	Reserved space.

Table 45. Entries elements in the FPZ4ABC-generated list

Name	Type	Description
Address	Fixed(64)	The address into the area mapped by the region memory registration (RMR) token.
Length	Fixed(32)	The length of the area, starting at address, to use.
Reserved	Fixed(32)	Reserved space.

FPZ4ABC

Table 46. Return and Reason Codes for the FPZ4ABC service

Hexadecimal Return Code	Reason Code	Meaning and Action
00	0000	Meaning: The call completed successfully. Action: None.
04	2000	Meaning: No zEDC devices are available. Inflate is completed in software when hardware is not available. Action: None.
08	0000	Meaning: No zEDC devices are available. Action: If zEDC devices are available to this system, perform diagnostics to determine the reason for the failure.
0C	0202	Meaning: One of the buffers had a length of 0, or the first word of a length was non-zero. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0203	Meaning: A failure occurred while accessing one of the provided scatter/gather buffers. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0206	Meaning: The output area was not large enough to complete the request. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0207	Meaning: The calling environment is invalid. The caller is either Problem State, non-zero key, or in XMEM mode. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0208	Meaning: The rendezvous token is invalid. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0209	Meaning: The region memory registration (RMR) token is invalid. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0221	Meaning: The header of the FPZ4ABC-generated list was not formed correctly. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0222	Meaning: Either zero or a number greater than the maximum supported was specified for the number of entries in the FPZ4ABC-generated list. Action: Check the calling program for a probable coding error. Correct the program and rerun it.

Table 46. Return and Reason Codes for the FPZ4ABC service (continued)

Hexadecimal Return Code	Reason Code	Meaning and Action
0C	0223	Meaning: A buffer in the scatter/gather list was not aligned properly. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0224	Meaning: The total length of the buffers in the scatter/gather list does not match the length in the parmlist. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	0225	Meaning: Scatter/gather was requested, but it was not enabled for this rendezvous token. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	1202	Meaning: An address range is not contained in the region denoted by the region memory registration (RMR) token. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	1203	Meaning: An unsupported operation was requested. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	1205	Meaning: An inflate request failed because of malformed data. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	2101	Meaning: An inflate request failed in software mode due to malformed input data. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
0C	2102	Meaning: Not enough space in the output buffer to process the request in software mode. Action: Check the calling program for a probable coding error. Correct the program and rerun it.
10	0301	Meaning: An internal component error occurred. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.
10	0304	Meaning: A rendezvous has not yet occurred for this address space. Action: Check that the application successfully called the FPZ4RZV service.
10	1203	Meaning: There are no zEDC devices available and either the request was a deflate request or software inflate was not enabled. Action: Check the calling program for a probable coding error. Correct the program and rerun it.

FPZ4ABC

Table 46. Return and Reason Codes for the FPZ4ABC service (continued)

Hexadecimal Return Code	Reason Code	Meaning and Action
10	1301	<p>Meaning: The request failed unexpectedly for an unknown reason.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

FPZ4URZ — Unrendezvous compression request

Description: The FPZ4URZ service removes the address space level information related to zEDC Express compression services. Any outstanding memory registrations are unregistered.

Table 47. Environment for the FPZ4URZ service

Environmental factor	Requirement
Minimum authorization:	Supervisor State
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	64-bit
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Table 48. Parameters for the FPZ4URZ service

Name	Type	Input/Output	Description
<i>ApplicationId</i>	Fixed(32)	Input	The application type to use. 0x01 is the application type for zEDC.
FPZ4URZ_options	Bit(64)	Input	There are no supported options for the FPZ4URZ service.
Rendezvous token	Char(16)	Input	The rendezvous token.
Return code	Fixed(31)	Output	The return code for the service.
Reason code	Fixed(32)	Output	The reason code for the service.

Table 49. Return and Reason Codes for the FPZ4URZ service

Hexadecimal Return Code	Reason Code	Meaning and Action
00	0000	<p>Meaning: The call completed successfully.</p> <p>Action: None.</p>
0C	0207	<p>Meaning: The calling environment is invalid. The caller is either Problem State, non-zero key, or in XMEM mode.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>
0C	0208	<p>Meaning: An invalid rendezvous token was passed.</p> <p>Action: Check the calling program for a probable coding error. Correct the program and rerun it.</p>

Table 49. Return and Reason Codes for the FPZ4URZ service (continued)

Hexadecimal Return Code	Reason Code	Meaning and Action
10	0301	Meaning: An internal error has caused recovery to be entered. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.
10	0304	Meaning: Compression services were not initialized. Action: Check the calling program for a probable coding error. Correct the program and rerun it.

Usage example of a System z authorized service

The following example uses the authorized services to perform compression using zEDC Express. **Note:** If zEDC Express adapters are not available, data is written to the destination uncompressed.

The FPZ4PRB service is called intermittently after the FPZ4ABC service returns to the application with a return code that indicates that all zEDC devices have left the configuration.

```

Call FPZ4RZV(AppId, RzvOptions, RzvUserId, RzvToken, RetCode, RsnCode)                /* Rendezvous with the compression
                                                                                   device driver (once per address
                                                                                   space) */

If RetCode = RcNoDevices Then                                                       /* If no devices available */
  NoDevices = ON                                                                    /* Indicate no devices */

Call FPZ4RMR(AppId, RmrOptions, RzvToken, InBuffer@, InBufferLen, 0, InBufKey, InRmrToken, /* Register the input buffer */
  RetCode, RsnCode)
Call FPZ4RMR(AppId, RmrOptions, RzvToken, OutBuffer@, OutBufferLen, 0, OutBufKey, OutRmrToken, /* Register the output buffer for
  RetCode, RsnCode)                                                                compressed data */

Do Until End of Data
  Read next block of data into InBuffer@

  If NoDevices = ON Then                                                            /* If no devices available */
    Call FPZ4PRB(AppId, Options, NumDevices, RetCode, RsnCode)                    /* Probe for new devices */

    If RetCode = RcOk Then                                                          /* If devices now available */
      NoDevices = OFF                                                            /* Indicate we have devices */
    Else                                                                            /* Else no devices */
      Write InBuffer                                                            /* Processed uncompressed data */
    End If

    If NoDevices = OFF Then                                                        /* If devices available */
      Call FPZ4ABC(RzvToken,
        InBuffer@, InBufferLen, InRmrToken,
        OutBuffer@, OutBufferLen, OutRmrToken,
        RetCode, RsnCode)                                                        /* Perform compression */

      If RetCode = RcOk Then                                                       /* If data was compressed */
        Write OutBuffer                                                         /* Process compressed data */
      Else If RetCode = RcNoDevices Then                                          /* If no devices available */
        NoDevices = ON                                                         /* Indicate no devices */
      End If
      Write InBuffer                                                            /* Process uncompressed data */
    End If
  End If
End Loop

Call FPZ4DMR(DmrOptions, RzvToken, InRmrToken, RetCode, RsnCode)
Call FPZ4DMR(DmrOptions, RzvToken, OutRmrToken, RetCode, RsnCode)

```

Chapter 15. Troubleshooting for zEDC

This topic explains troubleshooting techniques for zEDC.

RMF provides the following data for the System z accelerator device:

- Load current partition is putting on device
- Compression and decompression request rate and throughput
- Achieved compression ratio

See *z/OS RMF User's Guide* for the available options to specify on your Monitor I session for reporting on the System z compression accelerator.

Part 7. Other callable services

Chapter 16. IEAAFFN — Assign processor affinity for encryption or decryption

Call IEAAFFN when the only function performed by your program is to encrypt or decrypt data. Encryption and decryption take place on processors that have Integrated Cryptographic Features (ICRFs) associated with them. IEAAFFN assigns a program affinity to processors with an ICRF; that is, IEAAFFN makes sure the system runs your program on a processor that has an ICRF associated with it.

You do **not** have to use the IEAAFFN service to ensure the system runs a program on a processor with an ICRF; the system ensures that automatically. However, you can avoid some of the system overhead involved in the selection process by using the IEAAFFN service. IBM recommends that you use the service in programs whose **only** function is encryption or decryption.

Note: When you use this service to either establish or remove processor affinity for a program, the program permanently loses any processor affinity that the system programmer assigned to it in the SCHEDxx member of SYS1.PARMLIB.

Code the CALL following the syntax of the high level language you are using and specifying all parameters in the order shown.

CALL statement	Parameters
CALL IEAAFFN	(feature ,operation_type ,return_code)

The parameters are explained as follows:

feature

Specifies the feature required by your program. Specify CRYPTO to indicate an ICRF.

Define *feature* as character data of length 10. Pad the string on the right with 4 blanks.

,operation_type

Specifies the type of action you want to take. The types are:

GRANT

Establish affinity for the program to processors with an ICRF.

REMOVE

Remove affinity for the program to processors with an ICRF.

Note: After you issue a REMOVE request, the program has no processor affinity; it can run on any processor.

Define *operation_type* as character data of length 6. If you specify GRANT, pad the string on the right with 1 blank.

IEAAFFN

,return_code

When IEAAFFN completes, *return_code* contains the return code from the service. The return code value is also in register 15.

Define *return_code* as integer data of length 4. The return codes are explained under "Return codes."

Restrictions and limitations

Use the IEAAFFN service to request affinity to processors with an ICRF only for sections of a program that require an ICRF and not other features, such as a Vector Facility.

Requirements

Requirement	Details
Authorization:	Supervisor state or Problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	You can be either in cross memory mode or not
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	None held
Control parameters:	Must be in the primary address space

Return codes

When IEAAFFN returns control to your program, *return_code* and register 15 contain a return code. The following table identifies the return codes in hexadecimal and decimal (in parentheses), tells what each means, and recommends an action that you should take.

Table 50. IEAAFFN Return Codes

Return code	Meaning and Action
00000000 (0)	Meaning: The operation was successful. Action: None required.
00000004 (4)	Meaning: The program already had processor affinity assigned to it by the system programmer. The system replaces that affinity with the affinity you requested in this service. Action: None required.
0000000C (12)	Meaning: Your program was not running in task mode. Action: This service is not available to SRB mode programs. See the FEATURE= option on the SCHEDULE macro for the use of this function in SRB mode.
00000010 (16)	Meaning: The feature you specified was not a valid feature. Action: Specify a valid feature name.
00000014 (20)	Meaning: The operation type you specified was not valid. Action: Specify a valid operation type.
00000018 (24)	Meaning: The feature you specified is not installed on any of the processors in the system. Action: To the system programmer: See that the program runs on a system with the feature installed.

Table 50. IEAFFN Return Codes (continued)

Return code	Meaning and Action
000001C (28)	<p data-bbox="672 243 1052 275">Meaning: A system error has occurred.</p> <p data-bbox="672 296 1442 373">Action: To the system programmer: The error is recorded in LOGREC. Look for a record with a subcomponent of "IEAFFN CSS"; then call your IBM Support Center.</p>

IEAFFN

Chapter 17. CSRL16J — Transfer control to another routine

The CSRL16J service allows you to transfer control to another routine running under the same request block (RB) as the calling program. The CSRL16J service will transfer control with the contents of all 16 registers intact. When you transfer control to the other routine, use the CSRL16J service to:

- Define the entry characteristics and register contents for the target routine.
- Optionally free dynamic storage associated with the calling program.

When the service is successful, control transfers to the target routine. After the target routine runs, it can transfer control to any program running under the same request block (RB), including the calling program.

The CSRL16J service returns control to the calling program **only** when it cannot transfer control successfully to the target because of an error.

Defining the entry characteristics of the target routine

Specify the entry characteristics for the target in data area L16J, which forms the parameter list passed from the calling program to CSRL16J. Use the CSRYL16J mapping macro to see the format of the L16J parameter list. To build the L16J parameter list, first initialize the parameter list with zeroes and then fill in the desired fields. This ensures that all fields requiring zeroes are correct. You can specify the following characteristics for the target in L16J:

- Length of the L16J parameter list, L16JLENGTH field in mapping macro CSRYL16J.
- Contents of the general purpose registers (GPRs) 0-15, L16JGRS field in mapping macro CSRYL16J.
- Contents of the access registers (ARs) 0-15, L16JARS field in mapping macro CSRYL16J.
- PSW information for the target routine, field L16JPSW field in mapping macro CSRYL16J.
 - PSW address and AMODE
 - PSW ASC mode - primary or AR
 - PSW program mask
 - PSW condition code

Authorized callers, (callers in supervisor state, with PSW key 0-7, or with a PKM that allows any key 0-7) can specify:

- PSW state - problem or supervisor
- PSW key.

For unauthorized callers, the system uses the PSW state and key of the calling program for the target routine.

See *Principles of Operation* for more information about the contents of the PSW.

- Bit indicating whether or not you want to specify the contents of the access registers (ARs) for the target routine. This is the L16JPROCESSARS bit in mapping macro CSRYL16J.

Set the bit on if you want to specify the contents of the ARs. If you set the bit off, the system determines the contents of the ARs.

If the bit is set on when CSRL16J passes control to the target routine, the access registers (ARs) contain:

Register**Contents**

0-15 Specified by the caller

If the bit is set off when CSRL16J passes control to the target routine, the access registers (ARs) contain:

Register**Contents**

0-1 Do not contain any information for use by the routine

2-13 The contents are the same as they were when the caller issued the CSRL16J service.

14-15 Do not contain any information for use by the routine

Freeing dynamic storage associated with the caller

If the calling program has a dynamic storage area associated with it, you can specify that some or all of this storage area be freed before CSRL16J transfers control to the target. In the L16J parameter list, specify:

- The subpool of the area that you want the system to free. L16JSUBPOOL field in mapping macro CSRYL16J.
- The length, in bytes, of the dynamic storage area you want the system to free. L16JLENGHTHOFREE field in mapping macro CSRYL16J.
- The address of the dynamic storage area you want the system to free. L16JAREATOFREE field in mapping macro CSRYL16J.

Make sure that the address is on a double-word boundary. Otherwise the service ends with an abend code X'978'. See *z/OS MVS System Codes* for information on abend code X'978'.

The system frees the storage only when the CSRL16J service is successful.

Programming requirements

These are the requirements:

- The calling program must be in 31-bit addressing mode.
- Before you use the CSRL16J service, you must build a parameter list, L16J, to pass to the service. The parameter list includes the entry characteristics and environment for the target.

If you are coding in C/370, you can include the CSRLJC macro to provide declarations in the calling program for the L16J parameter area and return codes.

If you are coding in PL/I, you can include the CSRLJPLI macro to provide declarations for the return codes only. See Figure 20 on page 234 for the CSRLJPLI macro. Use the data area, mapped by the CSRYL16J mapping macro, as a model for the structure of your parameter list when coding in PL/I.

CSRLJC provides the following declarations for use in your C/370 program:

```

/*****
*           Type Definitions for User Specified Parameters           *
*****/

/* Type for user supplied L16J                                     */
typedef struct ??<

```

```

int Version;      /* Must be 0 */
int Length;      /* Initialize to CSRL16J_LENGTH */
int SubPool;     /* Subpool of storage to be freed */
union ??<
    char GRs??(64??); /* General registers */
    int GR??(16??);  /* General register 0-15 */
??> u1;
union ??<
    char ARs??(64??); /* Access registers */
    int AR??(16??);  /* Access register 0-15 */
??> u2;
union ??<
    char PSW??(8??); /* PSW: the processing will use the address,
                       AMODE, ASC mode, CC, and program mask. For a
                       supervisor state or PKM 0-7 or key 0-7
                       caller, it will use the state and key from
                       the PSW. Otherwise, it will set to caller
                       key and state. */
    struct ??<
        int PSWByte0to3 : 32; /* First 4 bytes */
        union ??<
            void *PSWAddr; /* Address and AMODE */
            struct ??<
                int PSWAmode : 1; /* AMODE */
                int Rsvd0 : 31;
            ??> s2;
        ??> u4;
    ??> s1;
??> u3;
union ??<
    struct ??<
        int Flags : 8; /* Flags */
        int Rsvd0 : 24; /* Reserved */
    ??> s3;
    struct ??<
        int ProcessARs : 1; /* If on, ARs will be processed. Otherwise
                             not. If not processed, ARs 0, 1, 14, and 15 are
                             unpredictable. ARs 2-13 are taken from the values
                             present when the service is entered. */
        int Rsvd0 : 31; /* Reserved */
    ??> s4;
??> u5;

void *AreaToFree; /* Address of area to free. If this is non-0
                  then the area will be freed using the subpool
                  specified in L16J.Subpool. This can be used
                  to free the caller's entire dynamic area if
                  so desired. When this option is specified, it
                  is necessary that the area begin on a
                  doubleword boundary. */
int LengthToFree; /* Length of area to free, in bytes. */
char Rsvd??(8??); /* Reserved */
??> L16J;
/*****
 * Fixed Service Parameter and Return Code Defines
 *****/

#define CSRL16J_LENGTH 168 /* Length of L16J */

/* Service Return Codes */
#define CSRL16J_OK 0
#define CSRL16J_BAD_VERSION 4
#define CSRL16J_BAD_AMODE 8
#define CSRL16J_BAD_RESERVED 12
#define CSRL16J_BAD_LENGTH 16
#define CSRL16J_BAD_PSW 24

```

```

/*****
 *          Function Prototypes for Service Routines          *
 *****/

extern void csr16j(
    L16J * __L16J,      /* Input - User supplied L16J block */
    int * __RC);       /* Output - Return code */

/*****

#endif

CSRLJPLI provides the following declarations for use in your PL/I program:

/*****
 *          Constants for Fixed Return Codes          *
 *****/

/* Load 16 and Jump Service Return Codes */

%DCL CSRL16J_OK FIXED;
%CSRL16J_OK          = 0;

%DCL CSRL16J_BAD_VERSION FIXED;
%CSRL16J_BAD_VERSION = 4;

%DCL CSRL16J_BAD_AMODE FIXED;
%CSRL16J_BAD_AMODE   = 8;

%DCL CSRL16J_BAD_RESERVED FIXED;
%CSRL16J_BAD_RESERVED = 12;

%DCL CSRL16J_BAD_LENGTH FIXED;
%CSRL16J_BAD_LENGTH   = 16;

%DCL CSRL16J_BAD_PSW FIXED;
%CSRL16J_BAD_PSW      = 24;

/*****
 *          Service Entry Declarations          *
 *****/

DCL CSRL16J ENTRY
    (CHAR(168), /* Input - L16J */
     FIXED BIN(31) /* Output - Return code */
     OPTIONS(INTER ASSEMBLER));

/* End of Load 16 and Jump Service Declares */

```

Figure 20. CSRLJPLI declarations for return codes for PL/I

Restrictions

None.

Performance implications

None.

Syntax diagram

Code the invocation following the syntax of the language you are using. Specify parameters in the order shown.

C/370 syntax

Code	Parameters
csrl16j	(&L16J ,&return_code)

PL/I syntax

Code	Parameters
CALL CSRL16J	(L16J ,return_code)

Parameters

The parameters are explained as follows:

L16J

Specifies a parameter list that the service uses to define the entry characteristics and environment for the target.

return_code

When the service completes, *return_code* contains the return code.

Return codes

If the CSRL16J service returns control to the caller, an error has occurred and the service was unable to transfer control to the target routine. In this case, the return code is always nonzero. When the service successfully transfers control to the target routine, the return code is zero.

Return codes from the CSRL16J service are as follows:

Table 51. CSRL16J Return Codes

Return Code (hexadecimal)	Meaning and Action
00	<p>Meaning: Successful completion. The calling program will never see this return code because it indicates that the target routine received control.</p> <p>Action: None.</p>
04	<p>Meaning: The value specified in the L16JVERSION field of the L16J data area was not a zero. The L16JVERSION field must contain a value of zero.</p> <p>Action: When you build the L16J data area, first zero the entire L16J data area and then fill in the required fields. This process ensures that all fields that must contain zeroes are correct.</p>

Table 51. CSRL16J Return Codes (continued)

Return Code (hexadecimal)	Meaning and Action
08	<p>Meaning: The calling program was not in 31-bit addressing mode, which is required.</p> <p>Action: Make sure the calling program is in 31-bit addressing mode.</p>
0C	<p>Meaning: One of the fields in the L16J data area that is reserved for IBM use contained a nonzero value. Any field reserved for IBM use must contain a value of zero.</p> <p>Action: When you build the L16J data area, first zero the entire L16J data area and then fill in the required fields. This process ensures that all fields that must contain zeroes are correct</p>
10	<p>Meaning: The value specified in field L16JLENGTH in the L16J data area was less than the actual length of the L16J.</p> <p>Action: Make sure that the value in the L16JLENGTH field reflects the actual length of the L16J data area.</p>
18	<p>Meaning: The PSW provided in field L16JPSW of the L16J data area specified an incorrect ASC mode.</p> <p>Action: In the L16JPSW field, specify either primary or AR ASC mode.</p>

Example

The following example, coded in C/370 uses CSRL16J to transfer control to a C/370 program. The target routine executes in the mode and with the register contents specified by the calling program in the L16J parameter list.

This example performs the following operations:

- Fills in L16J parameter list with PSW and execution mode data.
- Calls an assembler routine to obtain the current register contents of registers 0 through 13 and copies them to the L16J parameter list.
- Defines the contents of registers 14 and 15 for the target routine.
- Issues setjmp to allow return from the target routine.
- Invokes the C/370 function L16JPrg through CSRL16J.
- CSRL16J issues longjmp to return to caller and complete processing.

To use this example, you must also use the assembler program following the C/370 example.

C/370 example program

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <setjmp.h>
#include "CSRLJC.H"

#define FALSE 0
#define TRUE 1

/* REG0T013 is the assembler assist routine (below) to extract
   registers 0 through 13, for C/370 addressability */
#pragma linkage(REG0T013,0S)

int      rcode;
int      i;
```

```

unsigned int regs??(14??); /* Register save area */
jmp_buf      JumpBuffer; /* Buffer for setjmp/longjmp */
L16J         L16JParmArea; /* L16J parameter list structure */

/* Function prototype for function to be called via L16J */
void L16JPrg();

/* Invoke a C/370 function via L16J Callable Services */
main()
{
    /* Start by initializing the entire L16J parameter list */
    memset(&L16JParmArea,'\0',sizeof(L16J));

    /* The following fields were implicitly initialized to zero
       by the preceding statement:
       L16JParmArea.Version
       L16JParmArea.SubPool
       L16JParmArea.AreaToFree
       L16JParmArea.LengthToFree
       These field do not need to be explicitly set unless a value
       other than zero is required */

    /* Place parameter list length size into parameter list */
    L16JParmArea.Length = sizeof(L16J);

    /* Create a Problem State/Key 8 PSW */
    L16JParmArea.u3.s1.PSWByte0to3 = 0x078D1000;
    L16JParmArea.u3.s1.u4.PSWAddr = (void *) &L16JPrg;

    /* Mode data */
    L16JParmArea.u3.s1.u4.s2.PSWAmode = 1;
    L16JParmArea.u5.s4.ProcessARs = 1;

    /* Call assembler assist routine to obtain current register
       values */
    REGOT013(&regs);

    /* Place register values into parameter list */
    for (i=0;i<14;i++)
        L16JParmArea.u1.GR??(i??)= regs??(i??);

    /* Register 14 is not being used in this linkage, but we
       have set it to zero for this example */
    L16JParmArea.u1.GRAddr??(14??) = 0;

    /* Set register 15 for entry to routine */
    L16JParmArea.u1.GRAddr??(15??) = (void *) &L16JPrg;

    printf("L16JC - Call L16J to invoke L16JPrg\n");

    /* Use setjmp to allow return to this point in program. If
       setjmp is being called for the first time, invoke L16JPrg
       via L16J Callable Services. If returning from longjmp,
       skip call to L16J services and complete processing. */
    if (!setjmp(JumpBuffer))
    {
        csrl16j (&L16JParmArea,&rancode);

        /* Demonstrate use of L16J C/370 declares */
        switch (rancode)
        {
            /* Select on a particular return code value */
            case CSRL16J_BAD_PSW:
                printf("L16JC - L16J unsuccessful, bad PSW\n");
                break;
            /* Default error processing */
            default:

```

```

        printf("L16JC - L16J unsuccessful, RC = %d\n",rcode);
        break;
    }
}
printf("L16JC - Returned from L16JPrg\n");
}

/* The routine below receives control via L16J Callable Services.
   control is passed back to main via longjmp. */
void L16JPrg(void)
{
    printf("L16JC - L16JPrg got control\n");
    longjmp(JumpBuffer,1);
}

```

Assembler program for use with the C/370 example

To use this example you must assemble the following program and link it with the C/370 program.

```

SR0T013 CSECT
SR0T013 AMODE 31
SR0T013 RMODE ANY
*
* Assembler assist routine to save contents of registers 0 through 13
* to the area pointed to by register 1.
*
REG0T013 DS    0H
          ENTRY REG0T013
* Get address of the save area
          L     15,0(,1)
* Save registers 0 to 13
          STM  0,13,0(15)
* Return to the caller
          BR   14
          END  SR0T013

```

Chapter 18. CSRSI — System information service

Use the CSRSI service to retrieve system information. You can request information about the machine itself, the logical partition (LPAR) in which the machine is running, or the virtual machine hypervisor (VM) under which the system is running. The returned information is mapped by DSECTs in macro CSRSIIDF (for assembler language callers) or structures in header file CSRSIC (for C language callers).

The information available depends upon the availability of the Store System Information (STSI) instruction. When the STSI instruction is not available (which would be indicated by receiving the return code 4 (equate symbol CSRSI_ST SINOTAVAILABLE), only the SI00PCCACPID, SI00PCCACPUA, and SI00PCCACAFM fields within the returned infoarea are valid. When the STSI instruction is available, the validity of the returned infoarea depends upon the system:

- If the system is running neither under LPAR nor VM, then only the CSRSI_Request_V1CPC_Machine data are valid.
- If the system is running under a logical partition (LPAR), then both the CSRSI_Request_V1CPC_Machine data and CSRSI_Request_V2CPC_LPAR data are valid.
- If the system is running under a virtual machine hypervisor (VM), then all of the data (CSRSI_Request_V1CPC_Machine, CSRSI_Request_V2CPC_LPAR, and CSRSI_Request_V3CPC_VM) are valid.

You can request any or all of the information regardless of your system, and validity bits will indicate which returned areas are valid.

Description

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Problem state, key 8–15
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit when using the CALL CSRSI form (or csrsi in C), 31-bit when using an alternate form
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold a LOCAL lock, the CMS lock, or the CPU lock but is not required to hold any locks.

Programming requirements

The caller should include the CSRSIIDF macro to map the returned information and to provide equates for the service.

System information service (CSRSI)

Restrictions

None.

Input register information

The caller is not required by the system to set up any registers.

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

Syntax

CALL statement	Parameters
CALL CSRSI,	(Request ,Infoarealen ,Infoarea ,Returncode)

In C: the syntax is similar. You can use either of the following techniques to invoke the service:

1. CSRSI (Request,...Returncode);
 - When you use this technique, you must link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB.
2. CSRSI_byaddr (Request,...Returncode);
 - This second technique requires AMODE=31, and, before you issue the CALL, you must verify that the CSRSI service is available (in the CVT, both CVTOSEXT and CVTCSRSI bits are set on).

In Assembler: Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use either of the following techniques as an alternative to CALL CSRSI:

1. LOAD EP=CSRSI
Save the entry point address
...
Put the saved entry point address into R15
Issue CALL (15),...
2. L 15,X'10' Get CVT
L 15,X'220'(.15)
L 15,X'30'(.15) Get address of CSRSI
CALL (15),(...)
 - Both of these techniques require AMODE=31. If you use the second technique, before you issue the CALL, you must verify that the CSRSI service is available (in the CVT, both CVTOSEXT and CVTCSRSI bits are set on).

Parameters

Request

Supplied parameter:

- Type: Integer
- Length: Full word

Request identifies the type of system information to be returned. The field must contain a value that represents one or more of the possible request types. You add the values to create the full word. Do not specify a request more than once. The possible requests, and their meanings, are:

CSRSI_Request_V1CPC_Machine

The system is to return information about the machine.

CSRSI_Request_V2CPC_LPAR

The system is to return information about the logical partition (LPAR).

CSRSI_Request_V3CPC_VM

The system is to return information about the virtual machine (VM).

,Infoarealen

Supplied parameter:

- Type: Integer
- Range: X'1040', X'2040', X'3040', X'4040'
- Length: Full word

Infoarealen specifies the length of the infoarea parameter.

,Infoarea

Returned parameter:

- Type: Character
- Length: X'1040', X'2040', X'3040', X'4040' bytes

Infoarea is to contain the retrieved system information. (Infoarealen specifies the length of the provided area.) The infoarea must be of the proper length to hold the requested information. This length depends on the value of the Request parameter.

- When the Request parameter is CSRSI_Request_V1CPC_Machine, the returned infoarea is mapped by SIV1 and the infoarealen parameter must be X'2040'.
- When the Request parameter is CSRSI_Request_V1CPC_Machine plus CSRSI_Request_V2CPC_LPAR, the returned infoarea is mapped by SIV1V2 and the infoarealen parameter must be X'3040'.
- When the Request parameter is CSRSI_Request_V1CPC_Machine plus CSRSI_Request_V2CPC_LPAR plus CSRSI_Request_V3CPC_VM, the returned infoarea is mapped by SIV1V2V3 and the infoarealen parameter must be X'4040'.
- When the Request parameter is CSRSI_Request_V1CPC_Machine plus CSRSI_Request_V3CPC_VM, the returned infoarea is mapped by SIV1V3 and the infoarealen parameter must be X'3040'.
- When the Request parameter is CSRSI_Request_V2CPC_LPAR, the returned infoarea is mapped by SIV2 and the infoarealen parameter must be X'1040'.
- When the Request parameter is CSRSI_Request_V2CPC_LPAR plus CSRSI_Request_V3CPC_VM, the returned infoarea is mapped by SIV2V3 and the infoarealen parameter must be X'2040'.

System information service (CSRSI)

- When the Request parameter is CSRSI_Request_V3CPC_VM, the returned infoarea is mapped by SIV3 and the infoarealen parameter must be X'1040'.

,Returncode

Returned parameter:

- Type: Integer
- Length: Full word

Returncode contains the return code from the CSRSI service.

Return codes

When the CSRSI service returns control to the caller, Returncode contains the return code. To obtain the equates for the return codes:

- If you are coding in assembler, include mapping macro CSRSIIDF, described in *z/OS MVS Data Areas* in the z/OS Internet library (<http://www.ibm.com/systems/z/os/zos/bkserv/>).
- If you are coding in C, use include file CSRSIC.

The following table describes the return codes, shown in decimal.

Return Code and Equate Symbol	Meaning and Action
00 (0) CSRSI_SUCCESS	Meaning: The CSRSI service completed successfully. All information requested was returned. Action: Check the si00validityflags field to determine the validity of each returned area.
04 (4) CSRSI_ST SINOTAVAILABLE	Meaning: The CSRSI service completed successfully, but since the Store System Information (STSI) instruction was not available, only the SI00PCCACPID, SI00PCCACPUA, and SI00PCCACAFM fields are valid. Action: None required.
08 (8) CSRSI_SERVICE NOTAVAILABLE	Meaning: Environmental error: The CSRSI service is not available on this system. Action: Avoid calling the CSRSI service unless running on a system on which it is available.
12 (C) CSRSI_BADREQUEST	Meaning: User error: The request parameter did not specify a word formed from any combination of CSRSI_Request_V1CPC_Machine, CSRSI_Request_V2CPC_LPAR, and CSRSI_Request_V3CPC_VM. Action: Correct the parameter.
16 (10) CSRSI_BADINFOAREALEN	Meaning: User error: The Infoarealen parameter did not match the length of the area required to return the requested information. Action: Correct the parameter.
20 (14) CSRSI_BADLOCK	Meaning: User error: The service was called while holding a system lock other than CPU, LOCAL/CML, or CMS. Action: Avoid calling in this environment.

CSRSIC C/370 header file

For the C programmer, include file CSRSIC provides equates for return codes and data constants, such as Register service request types. To use CSRSIC, copy the file from SYS1.SAMPLIB to the appropriate local C library. Here are the contents of the file:

```
#ifndef __CSRSI

#define __CSRSI

/*****
 *      Type Definitions for User Specified Parameters      *
 *****/

/* Type for Request operand of CSRSI */
typedef int CSRSIRequest;

/* Type for InfoAreaLen operand of CSRSI */
typedef int CSRSIInfoAreaLen;

/* Type for Return Code */
typedef int CSRSIReturnCode;

/*****
 *      Function Prototypes for Service Routines      *
 *****/

#ifdef __cplusplus
extern "OS" ??<
#else
#pragma linkage(CSRSI_calltype,OS)
#endif
typedef void CSRSI_calltype(
    CSRSIRequest    __REQUEST, /* Input - request type */
    CSRSIInfoAreaLen __INFOAREALEN, /* Input - length of infoarea */
    void            * __INFOAREA, /* Input - info area */
    CSRSIReturnCode * __RC); /* Output - return code */

extern CSRSI_calltype csrsi;

#ifdef __cplusplus
??>
#endif

#ifndef __cplusplus
#define csrsi_byaddr(Request, Flen, Fptr, Rcptr) \
    \
    struct CSRSI_PSA* CSRSI_pagezero = 0; \
    CSRSI_pagezero->CSRSI_cvt->CSRSI_cvtcsrt->CSRSI_addr \
    (Request,Flen,Fptr,Rcptr); \
??>;
#endif
??>;
struct CSRSI_CSRT ??<
    unsigned char CSRSI_csrt_filler1 ??(48??);
    CSRSI_calltype* CSRSI_addr;

struct CSRSI_CVT ??<
    unsigned char CSRSI_cvt_filler1 ??(116??);
struct ??<
    int CSRSI_cvtddb_rsvd1 : 4; /* Not needed */
    int CSRSI_cvtosext : 1; /* If on, indicates that the
```

System information service (CSRSI)

```

                CVTOSLVL fields are valid                */
    int CSRSI_cvtddb_rsvd2 : 3;        /* Not needed        */
        ???> CSRSI_cvtddb;
    unsigned char CSRSI_cvt_filler2 ??(427??);
    struct CSRSI_CSRT * CSRSI_cvtcsrt;
    unsigned char CSRSI_cvt_filler3 ??(716??);
    unsigned char CSRSI_cvtoslvl0;
    unsigned char CSRSI_cvtoslvl1;
    unsigned char CSRSI_cvtoslvl2;
    unsigned char CSRSI_cvtoslvl3;
    struct ??<
        int CSRSI_cvtcsrsi : 1;        /* If on, indicates that the
                                        CSRSI service is available */
        int CSRSI_cvtoslvl1_rsvd1 : 7; /* Not needed        */
        ???> CSRSI_cvtoslvl4;
    unsigned char CSRSI_cvt_filler4 ??(11??);        /*
??>;

struct CSRSI_PSA ??<
    char CSRSI_psa_filler??(16??);
    struct CSRSI_CVT* CSRSI_cvt;
??>;

/* End of CSRSI Header */

#endif

/*****
/* sillvl represents the output for a V1 CPC when general CPC
/* information is requested
*****/

typedef struct ??<
    unsigned char _filler1??(32??); /* Reserved */
    unsigned char sillvlcpcmanufacturer??(16??); /*
                                                The 16-character (0-9
                                                or uppercase A-Z) EBCDIC name
                                                of the manufacturer of the V1
                                                CPC. The name is
                                                left-justified with trailing
                                                blank characters if necessary.
                                                */
    unsigned char sillvlcpcptype??(4??); /* The 4-character (0-9) EBCDIC
                                                type identifier of the V1 CPC.
                                                */
    unsigned char _filler2??(12??); /* Reserved */
    unsigned char sillvlcpcmodel??(16??); /* The 16-character (0-9 or
                                                uppercase A-Z) EBCDIC model
                                                identifier of the V1 CPC. The
                                                identifier is left-justified
                                                with trailing blank characters
                                                if necessary.
                                                */
    unsigned char sillvlcpcsequencecode??(16??); /*
                                                The 16-character (0-9
                                                or uppercase A-Z) EBCDIC
                                                sequence code of the V1 CPC.
                                                The sequence code is
                                                right-justified with leading
                                                EBCDIC zeroes if necessary.
                                                */
    unsigned char sillvlcpcplantofmanufacture??(4??); /* The 4-character
                                                (0-9 or uppercase A-Z) EBCDIC
                                                plant code that identifies the
                                                plant of manufacture for the
                                                V1 CPC. The plant code is

```

System information service (CSRSI)

```

left-justified with trailing
blank characters if necessary.
                                                                    */
unsigned char  _filler3??(3996??); /* Reserved                          */
??> si11v1;

/*****
/* si22v1 represents the output for a V1 CPC when information          */
/* is requested about the set of CPUs                                */
/*****

typedef struct ??<
unsigned char  _filler1??(32??); /* Reserved                          */
unsigned char  si22v1cpucapability??(4??); /*
An unsigned binary integer
that specifies the capability
of one of the CPUs contained
in the V1 CPC. It is used as
an indication of the
capability of the CPU relative
to the capability of other CPU
models.                                                                    */
unsigned int   si22v1totalcpucount          : 16; /* A 2-byte
unsigned integer
that specifies the
total number of CPUs contained
in the V1 CPC. This number
includes all CPUs in the
configured state, the standby
state, and the reserved state.
                                                                    */

unsigned int   si22v1configuredcpucount     : 16; /* A 2-byte
unsigned binary
integer that specifies
the total number of CPUs that
are in the configured state. A
CPU is in the configured state
when it is described in the
V1-CPC configuration
definition and is available to
be used to execute programs.
                                                                    */

unsigned int   si22v1standbycpucount        : 16; /* A 2-byte
unsigned integer
that specifies the
total number of CPUs that are
in the standby state. A CPU is
in the standby state when it
is described in the V1-CPC
configuration definition, is
not available to be used to
execute programs, but can be
used to execute programs by
issuing instructions to place
it in the configured state.
                                                                    */

unsigned int   si22v1reservedcpucount       : 16; /* A 2-byte
unsigned binary
integer that specifies
the total number of CPUs that
are in the reserved state. A
CPU is in the reserved state
when it is described in the
V1-CPC configuration
definition, is not available
to be used to execute
```

System information service (CSRSI)

```
programs, and cannot be made
available to be used to
execute programs by issuing
instructions to place it in
the configured state, but it
may be possible to place it in
the standby or configured
state through manually
initiated actions */

struct ??<
  unsigned char _si22v1mpcpucapaf??(??); /* Each individual
adjustment factor. */
  unsigned char _filler??(4050??);
??> si22v1mpcpucapafs;
??> si22v1;

#define si22v1mpcpucapaf si22v1mpcpucapafs._si22v1mpcpucapaf

/*****
/* si22v2 represents the output for a V2 CPC when information */
/* is requested about the set of CPUs */
/*****
typedef struct ??<
  unsigned char _filler1??(32??); /* Reserved */
  unsigned int si22v2cpcnumber : 16; /* A 2-byte
unsigned integer
which is the number of
this V2 CPC. This number
distinguishes this V2 CPC from
all other V2 CPCs provided by
the same logical-partition
hypervisor */
  unsigned char _filler2; /* Reserved */
struct ??<
  unsigned int _si22v21cpudedicated : 1; /*
When one, indicates that
one or more of the logical
CPUs for this V2 CPC are
provided using V1 CPUs that
are dedicated to this V2 CPC
and are not used to provide
logical CPUs for any other V2
CPCs. The number of logical
CPUs that are provided using
dedicated V1 CPUs is specified
by the dedicated-LCPU-count
value. When zero, bit 0
indicates that none of the
logical CPUs for this V2 CPC
are provided using V1 CPUs
that are dedicated to this V2
CPC. */
  unsigned int _si22v21cpushared : 1; /*
When one, indicates that
or more of the logical CPUs
for this V2 CPC are provided
using V1 CPUs that can be used
to provide logical CPUs for
other V2 CPCs. The number of
logical CPUs that are provided
using shared V1 CPUs is
specified by the
shared-LCPU-count value. When
zero, it indicates that none
of the logical CPUs for this
V2 CPC are provided using
```

System information service (CSRSI)

```
shared V1 CPUs. */

unsigned int _si22v2lcpuulimit : 1; /*
Utilization limit. When one,
indicates that the amount of
use of the V1-CPC CPUs that
are used to provide the
logical CPUs for this V2 CPC
is limited. When zero, it
indicates that the amount of
use of the V1-CPC CPUs that
are used to provide the
logical CPUs for this V2 CPC
is unlimited. */

unsigned int _filler3 : 5; /* Reserved */

??> si22v2lcpuc; /* Characteristics */
unsigned int si22v2total1cpucount : 16; /*
A 2-byte unsigned
integer that specifies the
total number of logical CPUs
that are provided for this V2
CPC. This number includes all
of the logical CPUs that are
in the configured state, the
standby state, and the
reserved state. */

unsigned int si22v2configured1cpucount : 16; /*
A 2-byte unsigned
binary integer that specifies
the total number of logical
CPUs for this V2 CPC that are
in the configured state. A
logical CPU is in the
configured state when it is
described in the V2-CPC
configuration definition and
is available to be used to
execute programs. */

unsigned int si22v2standby1cpucount : 16; /*
A 2-byte unsigned
binary integer that specifies
the total number of logical
CPUs that are in the standby
state. A logical CPU is in the
standby state when it is
described in the V2-CPC
configuration definition, is
not available to be used to
execute programs, but can be
used to execute programs by
issuing instructions to place
it in the configured state. */

unsigned int si22v2reserved1cpucount : 16; /*
A 2-byte unsigned
binary integer that specifies
the total number of logical
CPUs that are in the reserved
state. A logical CPU is in the
reserved state when it is
described in the V2-CPC
configuration definition, is
not available to be used to
execute programs, and cannot
```

System information service (CSRSI)

```

be made available to be used
to execute programs by issuing
instructions to place it in
the configured state, but it
may be possible to place it in
the standby or configured
state through manually
initiated actions */
unsigned char si22v2cpcname??(16??); /*
The 8-character EBCDIC name of
this V2 CPC. The name is
left-justified with trailing
blank characters if necessary. */
unsigned char si22v2cpccapabilityaf??(4??); /* Capability Adjustment
Factor (CAF). An unsigned
binary integer of 1000 or
less. The adjustment factor
specifies the amount of the
V1-CPC capability that is
allowed to be used for this V2
CPC by the logical-partition
hypervisor. The fraction of
V1-CPC capability is
determined by dividing the CAF
value by 1000. */
unsigned char _filler4??(16??); /* Reserved */
unsigned int si22v2dedicatedlcpucount : 16; /*
A 2-byte unsigned
binary integer that specifies
the number of configured-state
logical CPUs for this V2 CPC
that are provided using
dedicated V1 CPUs. (See the
description of bit
si22v2lcpudedicated.) */
unsigned int si22v2sharedlcpucount : 16; /*
A 2-byte unsigned
integer that specifies the
number of configured-state
logical CPUs for this V2 CPC
that are provided using shared
V1 CPUs. (See the description
of bit si22v2lcpushared.) */
unsigned char _filler5??(4012??); /* Reserved */
??> si22v2;

#define si22v2lcpudedicated si22v2lcpuc._si22v2lcpudedicated
#define si22v2lcpushared si22v2lcpuc._si22v2lcpushared
#define si22v2lcpuulimit si22v2lcpuc._si22v2lcpuulimit

/*****
/* si22v3db is a description block that comprises part of the */
/* si22v3 data. */
/*****

typedef struct ??<
unsigned char _filler1??(4??); /* Reserved */
unsigned int si22v3dbtotalcpucount : 16; /*
A 2-byte unsigned
binary integer that specifies
the total number of logical
CPUs that are provided for
this V3 CPC. This number
includes all of the logical
```

System information service (CSRSI)

```
CPUs that are in the
configured state, the standby
state, and the reserved state.
*/
unsigned int  si22v3dbconfiguredlcpucount      : 16; /*
A 2-byte unsigned
binary integer that specifies
the number of logical CPUs for
this V3 CPC that are in the
configured state. A logical
CPU is in the configured state
when it is described in the
V3-CPC configuration
definition and is available to
be used to execute programs.
*/

unsigned int  si22v3dbstandbylcpucount        : 16; /*
A 2-byte unsigned
binary integer that specifies
the number of logical CPUs for
this V3 CPC that are in the
standby state. A logical CPU
is in the standby state when
it is described in the V3-CPC
configuration definition, is
not available to be used to
execute programs, but can be
used to execute programs by
issuing instructions to place
it in the configured state.
*/

unsigned int  si22v3dbreservedlcpucount      : 16; /*
A 2-byte unsigned
binary integer that specifies
the number of logical CPUs for
this V3 CPC that are in the
reserved state. A logical CPU
is in the reserved state when
it is described in the V2-CPC
configuration definition, is
not available to be used to
execute programs, and cannot
be made available to be used
to execute programs by issuing
instructions to place it in
the configured state, but it
may be possible to place it in
the standby or configured
state through manually
initiated actions
*/

unsigned char si22v3dbcpcname??(8??); /* The 8-character EBCDIC name
of this V3 CPC. The name is
left-justified with trailing
blank characters if necessary.
*/

unsigned char si22v3dbcpccaf??(4??); /* A 4-byte unsigned binary
integer that specifies an
adjustment factor. The
adjustment factor specifies
the amount of the V1-CPC or
V2-CPC capability that is
allowed to be used for this V3
CPC by the
virtual-machine-hypervisor
program.
*/
```

System information service (CSRSI)

```

unsigned char si22v3dbvmhpidentifier?(16?); /* The 16-character
                                           EBCDIC identifier of the
                                           virtual-machine-hypervisor
                                           program that provides this V3
                                           CPC. (This identifier may
                                           include qualifiers such as
                                           version number and release
                                           level). The identifier is
                                           left-justified with trailing
                                           blank characters if necessary.
                                           */
unsigned char _filler2?(24?); /* Reserved */
??> si22v3db;
/*****
/* si22v3 represents the output for a V3 CPC when information
/* is requested about the set of CPUs
/*
/*****

typedef struct ??<
unsigned char _filler1?(28?); /* Reserved */
unsigned char _filler2?(3?); /* Reserved */
struct ??<
    unsigned int _filler3 : 4; /* Reserved */
    unsigned int _si22v3dbcount : 4; /*
        Description Block Count. A
        4-bit unsigned binary integer
        that indicates the number (up
        to 8) of V3-CPC description
        blocks that are stored in the
        si22v3dbe array.
        */
??> si22v3dbcountfield; /*
si22v3db si22v3dbe?(8?); /* Array of entries. Only the number
        indicated by si22v3dbcount
        are valid
        */
unsigned char _filler5?(3552?); /* Reserved */
??> si22v3;

#define si22v3dbcount si22v3dbcountfield._si22v3dbcount

/*****
/* SI00 represents the "starter" information. This structure is
/* part of the information returned on every CSRSI request.
/*
/*****

typedef struct ??<
char si00cpcvariety; /* SI00CPCVariety_V1CPC_MACHINE,
                    SI00CPCVariety_V2CPC_LPAR, or
                    SI00CPCVariety_V3CPC_VM */
struct ??<
    int _si00validsillv1 : 1; /* sillv1 was requested and
        the information returned is valid
        */
    int _si00validsi22v1 : 1; /* si22v2 was requested and
        the information returned is valid
        */
    int _si00validsi22v2 : 1; /* si22v2 was requested and
        the information returned is valid
        */
    int _si00validsi22v3 : 1; /* si22v3 was requested and
        the information returned is valid
        */
    int _filler1 : 4; /* Reserved */
??> si00validityfTags;
unsigned char _filler2?(2?); /* Reserved */
unsigned char si00pccacpid?(12?); /* PCCACPID value for this CPU

```

System information service (CSRSI)

```

unsigned char si00pccacpua??(2??); /* PCCACPUA value for this CPU */
unsigned char si00pccacafm??(2??); /* PCCACAFM value for this CPU */
unsigned char _filler3??(4??); /* Reserved */
unsigned char si00lastupdatetimestamp??(8??); /* Time of last STSI
update, via STCK */
unsigned char _filler4??(32??); /* Reserved */
??> si00;

#define si00validsi11v1 si00validityflags._si00validsi11v1
#define si00validsi22v1 si00validityflags._si00validsi22v1
#define si00validsi22v2 si00validityflags._si00validsi22v2
#define si00validsi22v3 si00validityflags._si00validsi22v3

/*****
/* siv1 represents the information returned when V1CPC_MACHINE */
/* data is requested */
*****/

typedef struct ??<
    si00 siv1si00; /* Area mapped by
struct si00 */
    siv1v1 siv1siv1v1; /* Area
mapped by struct siv1v1 */
    siv1v2 siv1siv1v2; /* Area
mapped by struct siv1v2 */
??> siv1;

/*****
/* siv1v2 represents the information returned when V1CPC_MACHINE */
/* data and V2CPC_LPAR data is requested */
*****/

typedef struct ??<
    si00 siv1v2si00; /* Area mapped by
by struct si00 */
    siv1v1 siv1v2siv1v1; /* Area
mapped by struct siv1v1 */
    siv1v2 siv1v2siv1v2; /* Area
mapped by struct siv1v2 */
    siv1v3 siv1v2siv1v3; /* Area
mapped by struct siv1v3 */
??> siv1v2;

/*****
/* siv1v2v3 represents the information returned when V1CPC_MACHINE */
/* data, V2CPC_LPAR data and V3CPC_VM data is requested */
*****/

typedef struct ??<
    si00 siv1v2v3si00; /* Area
mapped by struct si00 */
    siv1v1 siv1v2v3siv1v1; /* Area
mapped by struct siv1v1 */
    siv1v2 siv1v2v3siv1v2; /* Area
mapped by struct siv1v2 */
    siv1v3 siv1v2v3siv1v3; /* Area
mapped by struct siv1v3 */
??> siv1v2v3;

/*****
/* siv1v3 represents the information returned when V1CPC_MACHINE */
*****/

```

System information service (CSRSI)

```
/* data and V3CPC_VM data is requested */
/*****/

typedef struct ??<
    si00 sivlv3si00; /* Area mapped
                    by struct si00 */
    s11v1 sivlv3s11v1; /* Area
                    mapped by struct s11v1 */
    si22v1 sivlv3si22v1; /* Area
                    mapped by struct si22v1 */
    si22v3 sivlv3si22v3; /* Area
                    mapped by struct si22v3 */
??> sivlv3;

/*****/
/* siv2 represents the information returned when V2CPC_LPAR
/* data is requested */
/*****/

typedef struct ??<
    si00 siv2si00; /* Area mapped by
                  struct si00 */
    si22v2 siv2si22v2; /* Area
                    mapped by struct si22v2 */
??> siv2;

/*****/
/* siv2v3 represents the information returned when V2CPC_LPAR
/* and V3CPC_VM data is requested */
/*****/

typedef struct ??<
    si00 siv2v3si00; /* Area mapped
                    by struct si00 */
    si22v2 siv2v3si22v2; /* Area
                    mapped by struct si22v2 */
    si22v3 siv2v3si22v3; /* Area
                    mapped by struct si22v3 */
??> siv2v3;

/*****/
/* siv3 represents the information returned when V3CPC_VM
/* data is requested */
/*****/

typedef struct ??<
    si00 siv3si00; /* Area mapped by
                  struct si00 */
    si22v3 siv3si22v3; /* Area
                    mapped by struct si22v3 */
??> siv3;

/*****/
* Fixed Service Parameter and Return Code Defines *
/*****/

/* SI00 Constants */

#define SI00CPCVARIETY_V1CPC_MACHINE 1
#define SI00CPCVARIETY_V2CPC_LPAR 2
#define SI00CPCVARIETY_V3CPC_VM 3

/* CSRSI Constants */

#define CSRSI_REQUEST_V1CPC_MACHINE 1
#define CSRSI_REQUEST_V2CPC_LPAR 2
```

System information service (CSRSI)

```
#define CSRSI_REQUEST_V3CPC_VM      4

/* CSRSI Return codes */

#define CSRSI_SUCCESS                0
#define CSRSI_STSNOTAVAILABLE        4
#define CSRSI_SERVICENOTAVAILABLE    8
#define CSRSI_BADREQUEST             12
#define CSRSI_BADINFOAREALEN        16
#define CSRSI_BADLOCK                20
```

System information service (CSRSI)

Part 8. Base Control Program internal interface (BCPii) services

Chapter 19. Base Control Program internal interface (BCPii)

IBM provides support within z/OS that allows authorized applications to query, change, and perform operational procedures against the installed System z hardware base through a set of application program interfaces. These applications can access the System z hardware that the application is running on and extend their reach to other System z processors within the attached process control (Hardware Management Console) network.

Using the Base Control Program internal interface (BCPii), an authorized z/OS application can perform the following actions:

- Obtain the System z topology of the current interconnected Central Processor Complexes (CPCs) as well as the images, capacity records, activation profiles, and user-defined image groups defined on a particular CPC.
- Query CPC, image (LPAR), capacity record, activation profile, and user-defined image group information.
- Set various configuration values related to CPC, image and activation profiles.
- Issue commands against CPCs, images (LPARs), and user-defined image groups to perform minor or even significant hardware- and software-related functions.
- Listen for various hardware and software events that might take place on various CPCs and images throughout the HMC-connected network.

Communication to the Support Element (SE) / Hardware Management Console (HMC) using BCPii is done completely within the base operating system and therefore does not require communication on an IP network (intranet) for connectivity, providing complete isolation of your System z hardware communication from any other network traffic within the intranet/internet.

Calls using the BCPii Application Programming Interfaces (APIs) can be made from the C, the REXX, or the assembler programming languages. See “Syntax, linkage and programming considerations” on page 271 for an explanation of how the APIs are called and see the explanation of each service for the syntax for each of the BCPii APIs.

BCPii setup and installation

Before an installation begins to issue BCPii APIs, a series of setup and installation steps must be performed. A summary of these steps follows. For additional details on each of these steps, see the supporting documentation that explains how each of these steps is accomplished:

1. Configure the local Support Element (SE) to support BCPii:
 - a. Check the levels of hardware that BCPii supports.
 - b. Enable cross-partition authority for each image (LPAR) that you want to grant BCPii access.
 - c. Define an uppercase BCPii SNMP community name on the SE.

See “Setting up connectivity to the support element” on page 258 for details.

2. Authorize an application to use BCPii, including authority to specific resources (such as CPCs, images and capacity records):
 - a. Check that the BCPii application is program-authorized.

- b. Check that the BCPii application has general authority to use BCPii.
- c. Authorize the BCPii application to access the particular resource that requires BCPii service.
- d. Define an uppercase BCPii SNMP community name in the security product for each CPC as it was defined on the SE. Use the APPLDATA field with the CPC profile definition to associate a BCPii SNMP community name with a particular CPC.

These steps enable communication to the local CPC and allows the BCPii address space to initialize. See “Setting up authority to use BCPii” on page 262 for details.

3. Configure the BCPii address space. See “BCPii configuration considerations” on page 264 for details.
4. If the caller is running in a z/OS UNIX System Services environment, set up the notification mechanism to allow hardware and software events to be propagated to the z/OS UNIX application. See “Setting up event notification for BCPii z/OS UNIX applications” on page 266 for details.
5. If the installation allows TSO/E users to have access to the BCPii APIs using REXX, see “Setting up an environment to run BCPii TSO/E REXX execs” on page 268.

After you have activated the BCPii address space, you need to know how to control the address space. See “BCPii startup and shutdown” on page 269 for details.

Figure 21 shows the steps needed to setup and install BCPii.

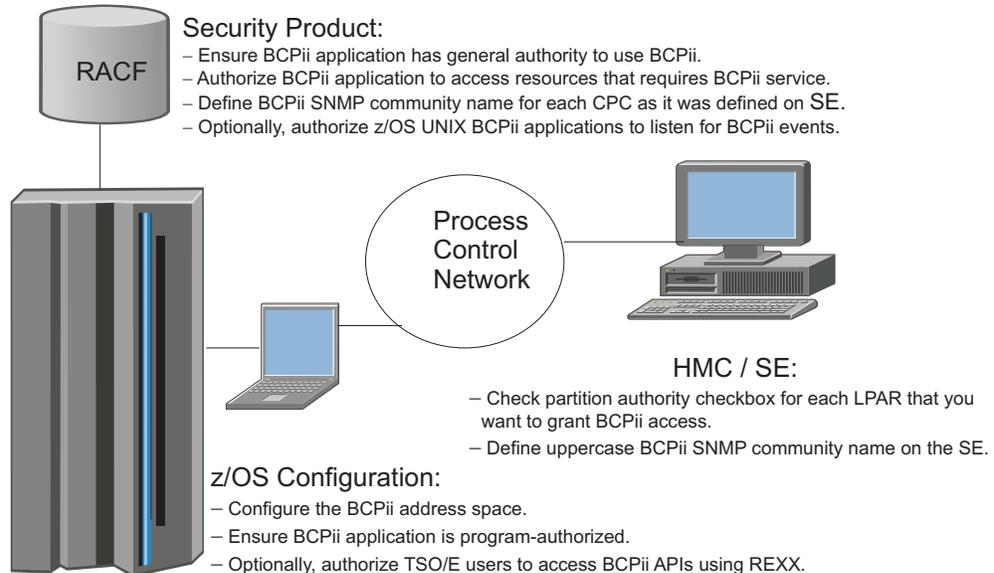


Figure 21. BCPii setup and installation steps

Setting up connectivity to the support element

BCPii uses a low-level operating system connection to establish communication between an authorized application running on a z/OS image (LPAR) and the Support Element (SE) associated with the Central Processor Complex (CPC) that

contains this z/OS image. You must configure the support element to permit these BCPii communications if BCPii services are required to be available by your installation.

Note: In order to customize the API settings controls on the SE, your userid must have administrator rights to access these panels.

Levels of hardware that BCPii supports

The HWIBCPii address space, which supports the issuing of BCPii APIs from a z/OS image, will run on any hardware that supports a level of the z/OS operating system in which BCPii is included. However, there will be some reduced BCPii functionality when a BCPii request targets a system that is not running on a zEnterprise[®] machine. The BCPii restrictions increase the further downlevel the hardware is from a zEnterprise machine. To run with the fewest functionality restrictions possible, make sure the recommended microcode levels are installed for that SE, HMC and LPAR hardware.

BCPii applications might need to perform hardware or software functions on CPCs other than the CPC on which the application is running. Such requests can be targeted to other System z[®] hardware at a lower or higher hardware level than the local CPC, provided that these hardware levels are supported to coexist with the local CPC level.

The HWICMD service is only allowed to be targeted to at least a System z9[®] hardware level running on a particular microcode level. BCPii rejects the targeting of this service to any System z hardware level earlier than System z9. See “HWICMD — Issue a BCPii hardware management command” on page 282 for further information.

Consult Table 52 to determine the minimum level of microcode required to run BCPii on a specific hardware level.

Table 52. Minimum BCPii microcode levels by SE hardware level

SE hardware level	Minimum microcode level
IBM System z9 Driver 67	MCL 258 in the G40965 (SE-SYSTEM) EC stream
IBM System z10 [®] Driver 79	MCL 163 in the N24409 (SE-SYSTEM) EC stream
IBM zEnterprise 196	MCL 220 in the N29802 (SE-SYSTEM) EC stream
IBM zEnterprise EC12	Any level

Consult Table 53 to determine the minimum level of microcode required to run BCPii on a specific HMC level.

Table 53. Minimum BCPii microcode levels by HMC level

HMC level	Minimum microcode level
IBM System z9 Driver 67	MCL 158 in the G40969 (HMC-SYSTEM) EC stream
IBM System z10 Driver 79	MCL 034 in the N24415 (HMC-SYSTEM) EC stream
IBM zEnterprise 196	Any level
IBM zEnterprise EC12	Any level

Consult Table 54 on page 260 to determine the minimum level of microcode required to run BCPii on a specific LPAR level.

Table 54. Minimum BCPii microcode levels by LPAR level

LPAR level	Minimum microcode level
IBM System z9 Driver 67	MCL 008 in the G40954 (LPAR) EC stream
IBM System z10 Driver 79	MCL 002 in the N24404 (LPAR) EC stream
IBM zEnterprise 196	Any level
IBM zEnterprise EC12	Any level

Each version of hardware has subtle or sometimes significant changes in the way information is displayed and saved in the support element. The examples serve as a guide only to where the actual definitions that need to be modified are located within the support element configuration windows.

Enable BCPii communications on the support element

You need to enable cross-partition authority on the support element to allow the support element to accept the BCPii APIs flowing from the user application through the HWIBCPii address space. This setting controls whether a logical partition can issue a subset of control program instructions to other logical partitions activated on the same CPC.

Note: This setting must be selected on the local SE associated with the CPC of the image that the z/OS BCPii application is running on. It must also be selected for any other system for which BCPii communication is required.

To change this setting, perform the following steps on the HMC:

1. Select the CPC that is required.
2. Open Single Object Operations.
3. Open the CPC Operational Customization task list.
4. Highlight the CPC icon.
5. Open the Change LPAR Security task, and the Change Logical Partition Security window displays.
6. Check the cross-partition authority checkbox for each image (LPAR) that you want to grant BCPii access. At a minimum, the image (LPAR) the BCPii address space is running needs to have this authority activated.
7. Select Save and Change.

See the HMC book and *System z9 Support Element Operations Guide* and *System z10 Support Element Operations Guide* for more information regarding changing the support element settings.

Failure to set this properly on the local SE associated with the image of z/OS that is running BCPii results in a severe BCPii address space initialization failure. You cannot start the address space and will receive communications error X'101' with a reason code of X'D4'. Failure to set this up properly on remote SEs to which you want to connect results in the same return code and reason code on the HWICONN service call.

Note: Make the same updates to all CPCs that you want BCPii to communicate with and not just the CPC from which the BCPii application is going to run on.

Define the BCPii community name on the support element

BCPii uses an SNMP community name to provide a level of security between the z/OS image that is executing the BCPii service and the support element itself.

An SNMP community is a logical relationship between an SNMP agent and an SNMP manager. The community has a name, and all members of a community have the same access privileges: they are either read-only (members can view configuration and performance information) or read-write (members can view configuration and performance information, and also change the configuration).

To add the BCPii community name definition to the SE configuration, perform the following steps on the HMC:

1. Select the CPC that is required.
2. Open Single Object Operations.
3. Select the Console Actions view.
4. Select Support Element Settings.
5. Open the Customize API Settings.
6. Check the Enable SNMP APIs checkbox.
7. Consider checking the "Allow capacity change API requests" checkbox on a z10 or higher operation system if the installation is to allow a BCPii application to perform temporary capacity upgrades.
8. Make sure that the SNMP agent parameters are blank.
9. Add a BCPii community name. Click on Add. When a window is prompted, fill in the following fields:

Name The actual SNMP community name. This value is a 1- to 16-character alphanumeric field. Only uppercase letters and numbers are allowed. Because of restrictions with the security products on z/OS, the BCPii SNMP community name must not contain any lowercase characters. See "Community name defined in the security product for each CPC" on page 263 for more information about the SNMP community name.

Address

For BCPii, this address (sometimes referred to as a loop-back address) must be 127.0.0.1.

Network mask/Prefix

255.255.255.255.

Access Type

Read/write

10. Save the changes.

See *System z9 Support Element Operations Guide* and *System z10 Support Element Operations Guide* for more information regarding changing the support element settings.

Failure to set this properly on the local SE associated with the image of z/OS that is running BCPii results in a severe BCPii failure and you cannot start the address space. Message HWI022I might be issued if the community name defined on the support element for the local CPC does not match the definition in the security product for the local CPC. See "Community name defined in the security product for each CPC" on page 263 for more information.

Note: Make the same updates to all CPCs that you want BCPii to communicate with.

Setting up authority to use BCPii

Given the nature of the BCPii APIs and the capabilities of a BCPii application to potentially modify vital hardware resources, a number of authority validations are performed for each BCPii requestor. A BCPii application needs to have program authority, general security product authority to be able to issue BCPii commands, authority to the particular resource that the application is trying to access, and a community name defined in the security product for each CPC to which communication is required.

Program authority

BCPii applications must be program-authorized, meaning that one of the following must be true of the application:

- Running in supervisor state.
- Running in an authorized key with PSW key mask (PKM) between 0 and 7.
- Residing in an APF-authorized library.

General security product authority

A BCPii application needs to have general authority to use BCPii. The profile HWI.APPLNAME.HWISERV in the FACILITY resource class controls which applications can use BCPii services. The security administrator must give at least read authority to this resource, in addition to granting authority to any specific resource that the application is attempting to access. In addition, BCPii requires that the FACILITY class to be RACLIST-specified. The RACF syntax is as follows:

```
RDEFINE FACILITY HWI.APPLNAME.HWISERV UACC(NONE)
PERMIT HWI.APPLNAME.HWISERV CLASS(FACILITY) ID(userid) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

This RACF example allows user JOE to use BCPii services in general:

```
RDEFINE FACILITY HWI.APPLNAME.HWISERV UACC(NONE)
PERMIT HWI.APPLNAME.HWISERV CLASS(FACILITY) ID(JOE) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Generic definitions may be created instead of specific users if the installation does not have specific definitions for every user.

This RACF example defines user IDs BCPii and HWIstart to the security product:

```
ADDUSER BCPii DFLTGRP(SYS1)
RDEFINE STARTED BCPii.** STDATA(USER(BCPii) GROUP(SYS1))
ADDUSER HWIstart DFLTGRP(SYS1)
RDEFINE STARTED HWIstart.** STDATA(USER(BCPii) GROUP(SYS1))
SETROPTS RACLIST(STARTED) REFRESH
```

Authority to the particular resource

A BCPii application needs to have authority to the particular resource that it is trying to access. That particular resource can be the CPC itself, an image (LPAR) on a particular CPC, or a particular capacity record on a particular CPC. BCPii needs a profile defined in the FACILITY resource class that represents the target of the particular BCPii request. The profile name required to be defined depends on the type of the particular resource required.

Request Type	FACILITY Class Profile Required
CPC	HWI.TARGET.netid.nau where netid.nau represents the 3- to 17-character SNA name of the particular CPC.

Request Type	FACILITY Class Profile Required
Image	HWI.TARGET. <i>netid.nau.imagename</i> where <i>netid.nau</i> represents the 3- to 17-character SNA name of the particular CPC and <i>imagename</i> represents the 1- to 8-character LPAR name.
Capacity record	HWI.CAPREC. <i>netid.nau.caprec</i> where <i>netid.nau</i> represents the 3- to 17-character SNA name of the particular CPC and <i>caprec</i> represents an 8-character capacity record name.
Activation profiles	HWI.TARGET. <i>netid.nau</i> where <i>netid.nau</i> represents the 3- to 17-character SNA name of the particular CPC the activation profile is defined.
User-defined image groups	HWI.TARGET. <i>netid.nau</i> where <i>netid.nau</i> represents the 3- to 17-character SNA name of the particular CPC the user-defined image group is defined.

Note: For compatibility with security products, BCPii automatically transforms the following names to all uppercase characters: CPC names (including the local CPC name represented by '*'), image names, and capacity record names specified on the HWICONN service.

The access level required for the particular profile depends on the service that the BCPii application attempts to issue. See the BCPii API documentation in this chapter for specifics regarding the minimum access level required for each BCPii API service. The RACF syntax is as follows:

```
RDEFINE FACILITY HWI.TARGET.netid.nau UACC(NONE) APPLDATA('uppercasecommunityname')
PERMIT HWI.TARGET.netid.nau CLASS(FACILITY) ID(userid) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

where *netid.nau* represents the 3 to 17 character SNA name of the CPC.

This RACF example allows user JOE to have Connect, Event, List, and Query access to CPC NET1.CPC001, using community name XYZ123. See "Community name defined in the security product for each CPC" for more details.

```
RDEFINE FACILITY HWI.TARGET.NET1.CPC001 UACC(NONE) APPLDATA('XYZ123')
PERMIT HWI.TARGET.NET1.CPC001 CLASS(FACILITY) ID(JOE) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

This RACF example grants user JOE with Command, Connect, Event, List, Query, and Set access to any image (LPAR) on NET1.CPC001:

```
RDEFINE FACILITY HWI.TARGET.NET1.CPC001.* UACC(NONE)
PERMIT HWI.TARGET.NET1.CPC001.* CLASS(FACILITY) ID(JOE) ACCESS(ALTER)
SETROPTS RACLIST(FACILITY) REFRESH
```

Community name defined in the security product for each CPC

BCPii uses an SNMP community name to provide a minimal level of security between the z/OS image executing the BCPii service and the support element itself.

An SNMP community name is associated with a particular CPC. The same SNMP community name that was defined in the support element configuration for a particular CPC also must be defined in the security product for each CPC to which communication is required. This community name definition is extracted from the security product by BCPii and propagated to the support element. The support element validates that the community name passed by BCPii is correct before

proceeding with the request. See *Define the BCPii community name on the Support Element* for information about how to define the community name on the SE or how to obtain the already-defined name.

To define the BCPii community name in the security product, use the APPLDATA field with the CPC profile definition to associate a community name with a particular CPC. The RACF syntax is as follows:

```
RALTER FACILITY HWI.TARGET.netid.nau APPLDATA('uppercasecommunityname')
SETROPTS RACLIST(FACILITY) REFRESH
```

where *netid.nau* represents the 3 to 17 character SNA name of the CPC.

The APPLDATA field for the BCPii community name contains a 1– to 16–character alphanumeric field. Only uppercase letters and numbers are allowed. Because of restrictions with the security products on z/OS, the BCPii SNMP community name must not contain any lowercase characters.

This RACF example assigns a BCPii community name of XYZ123 to an existing CPC definition for CPC name NET1.CPC001:

```
RALTER FACILITY HWI.TARGET.NET1.CPC001 APPLDATA('XYZ123')
SETROPTS RACLIST(FACILITY) REFRESH
```

Note: A community name definition must be defined for at least the local CPC. Otherwise, BCPii cannot continue with initialization of its address space and BCPii services are not available. This is accompanied by message HWI022I.

BCPii configuration considerations

The BCPii address space is the bridge between a z/OS application and the support element. The address space can perform the following steps:

- Manage all application connections.
- Builds and receive all internal communication requests to the SE.
- Provide an infrastructure for storage required by callers and by the transport communicating with the SE.
- Provide diagnostic capabilities to help with BCPii problem determination.
- Provide security authentication of requests.

The BCPii address space is mandatory for any BCPii API request. The system attempts to start the HWIBCPii address space during IPL.

BCPii requires the *high-level-qualifier.SCEERUN2* and *high-level-qualifier.SCEERUN* data sets to be in the link list concatenation. IBM specifies these data sets in the default link list members (PROGxx) in z/OS 1.10 and higher. BCPii also requires the *high-level-qualifier.SCEERUN2* and *high-level-qualifier.SCEERUN* data sets to be APF authorized. Failure to have these two data sets in the link list or APF authorized results in BCPii not being able to be started, accompanied by error message HWI009I that indicates that BCPii could not load a required Language Environment part.

BCPii also includes a parmlib member into SYS1.PARMLIB for default CTRACE settings (CTIHWI00) when BCPii initializes. See *z/OS MVS Diagnosis: Tools and Service Aids* for further information regarding CTRACE settings in BCPii.

BCPii writes SMF record 106 (X'6A') for certain API invocations. An SMFPRMxx parmlib member must be configured and activated in order to capture these

records. See “SMF recording in BCPii” on page 270 or *z/OS MVS System Management Facilities (SMF)* for more information about how BCPii uses SMF.

Considerations for Language Environment runtime options

z/OS BCPii uses z/OS Language Environment® (LE) to fulfill an API request from the caller. It creates an environment that usually does not conflict with an installation's default runtime option settings. However, specifying the NONOVR attribute to customize the LE runtime options, either via the CEEPRMxx parmlib member or via the SETCEE command, can result in incompatible settings that can lead to incorrect behavior by BCPii, including an abend when BCPii initializes and attempts to open the SYSOUT data set.

When you specify the NONOVR attribute for a runtime option, that runtime option cannot be overridden later, including by a later specification in the same parmlib member or by a subsequent SETCEE command.

BCPii requires the following LE runtime options and will attempt to override them:

```
TRAP(ON,NOSPIE)
POSIX(OFF)
ALL31(ON)
STACK(,ANY)
DYNDUMP(*USERID,NODYNAMIC,TDUMP)
MSGFILE(SYSOUT,,,ENQ)
TERMTHDACT(UADUMP,,256)
```

Specifying the NONOVR attribute for any of these options can result in the following error message when BCPii starts and attempts to override the options that can no longer be overridden:

```
CEE3768I The system default for the run-time option option could not be overridden.
```

IBM recommends that you *not* specify the NONOVR attribute for these options in order to allow BCPii's SYSOUT initialization to complete successfully and, in general, to allow BCPii to behave properly.

Dynamic modification of CPC names

An installation that implements a dynamic CPC name change on a CPC which either has BCPii active on one or more of its z/OS images or is a CPC that is targeted by other images on remote CPCs in the HMC network must review the following considerations before performing the name change.

BCPii does not currently provide support for changing the name of a CPC with ACTIVE images. Therefore, the only way BCPii can “learn” of the name change is to recycle the HWIBCPii address space. Evaluate the following considerations to determine which applies to your installation.

Applications that dynamically learn of new CPC names when BCPii is restarted

If *all* BCPii applications on a z/OS image will learn of the new CPC name dynamically when BCPii is restarted, perform the following steps:

1. Review security definitions impacted by CPC name change. For details, see “Security definition considerations for CPC name changes” on page 266.
2. Stop BCPii. For details, see “Ending the HWIBCPii address space” on page 269.

3. Change the CPC name.
4. Make the required security definition changes if necessary.
5. Restart BCPii. For details, see “Restarting the HWIBCPii address space” on page 269.

Applications that do not dynamically learn of new CPC names when BCPii is restarted

If any BCPii application maintains a persistent copy of the changed CPC name (whether it is the local name or the remote name), regardless of BCPii being restarted, that application will also need to be restarted. Perform the following steps for each such application:

1. Review security definitions impacted by the CPC name change. For details, see “Security definition considerations for CPC name changes.”
2. Stop the BCPii application.
3. Stop BCPii. For details, see “Ending the HWIBCPii address space” on page 269.
4. Change the CPC name.
5. Make any security definition changes that might be necessary.
6. Restart BCPii. For details, see “Restarting the HWIBCPii address space” on page 269.
7. Restart the BCPii application.

System Status Detect partitioning protocol

In the case of the XCF System Status Detect (SSD) partitioning protocol, while SSD can be recycled by using the SETXCF command, SSD does not pick up the CPC name change through this mechanism. Therefore, an IPL is necessary in order for XCF to learn of the CPC name change. If the installation has SSD active, it is necessary to perform a rolling IPL of all images in the sysplex that utilize SSD on the CPC whose name was changed. This allows XCF to pick up the name change on all systems in the sysplex that use SSD.

Security definition considerations for CPC name changes

It is necessary to review the security definitions for *HWI.TARGET.netid.nau*, *HWI.TARGET.netid.nau.imagename*, and *HWI.CAPREC.netid.nau.caprecname* profiles whenever a CPC name change is implemented to ensure the proper security configuration is in effect. Review any profiles that may need to be modified before the name change takes place. Ensure any changes to security profiles are made prior to the restart of BCPii. See “Setting up authority to use BCPii” on page 262 for general information on configuring the appropriate security definitions.

Setting up event notification for BCPii z/OS UNIX applications

Applications running in a started procedure, batch, TSO or other non z/OS UNIX environment can use the HWIEVENT service and provide their own ENF exit that receives control when the application-requested events occur on the target CPC or image.

Applications running in a z/OS UNIX environment do not have normal ENF exit processing capabilities available and cannot readily listen for ENF signals. The Common Event Adapter (CEA) address space allows z/OS UNIX applications to be able to receive such event notifications. BCPii provides several services that use the CEA functionality to deliver these same events to z/OS UNIX callers. See the

documentation for the z/OS UNIX-only services of BCPii (“HWIBeginEventDelivery — Begin delivery of BCPii event notifications” on page 399, “HWIEndEventDelivery — End delivery of BCPii event notifications” on page 403, “HWIManageEvents — Manage the list of BCPii events” on page 406, and “HWIGetEvent — Retrieve outstanding BCPii event notifications” on page 410) for details about the services a z/OS UNIX application can use to receive event notification.

The use of the CEA address space by BCPii requires some minor CEA setup before z/OS UNIX-only services of BCPii can work properly.

CEA address space setup

The Common Event Adapter (CEA) address space must be active to allow the z/OS UNIX-only services of BCPii to operate. CEA has two modes of operation: minimum or full-function mode. If the z/OS UNIX-only services of BCPii are required to be available, CEA must be running in full-function mode. To activate full-function mode, a set of security product definitions are required. See *z/OS Planning for Installation* for more information about how to configure Common Event Adapter for full-function mode.

CEA, like BCPii, starts as part of a system IPL. It can be stopped and restarted as well. See *z/OS Planning for Installation* for more information.

CEA ENF security configuration

A z/OS UNIX BCPii application must be granted authority to listen to ENF68 events. With the CEA ENF controls, it is also possible to fine-tune which BCPii events a user is allowed to listen to.

This RACF example gives generic authority to the user id associated with a z/OS UNIX application authority to listen to any BCPii event:

```
AU user_id OMVS(Uid(n))
SETROPTS GENERIC(SERVAUTH)
RDEFINE SERVAUTH CEA.CONNECT UACC(NONE)
RDEFINE SERVAUTH CEA.SUBSCRIBE.ENF_0068* UACC(NONE)
PERMIT CEA.CONNECT CLASS(SERVAUTH) ID(user_id) ACCESS(READ)
PERMIT CEA.SUBSCRIBE.ENF_0068* CLASS(SERVAUTH) ID(user_id) ACCESS(READ)
SETROPTS RACLIST(SERVAUTH) REFRESH
```

To give specific authority to only certain BCPii events, use the event qualifier as part of the profile name. The event qualifier maps to the event mask for ENF68 in the ENFREQ documentation in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*. Hardware events are in the form ‘03xx00yy’ where *xx* is the event source (‘01’x = CPC, and ‘02’x =image) and *yy* denotes the particular event.

This RACF example allows user JOE authority to only receive events related to CPC command responses (CmdResp = ‘01’x):

```
AU JOE OMVS(Uid(5))
RDEFINE SERVAUTH CEA.CONNECT UACC(NONE)
RDEFINE SERVAUTH CEA.SUBSCRIBE.ENF_006803010001 UACC(NONE)
PERMIT CEA.CONNECT CLASS(SERVAUTH) ID(JOE) ACCESS(READ)
PERMIT CEA.SUBSCRIBE.ENF_006803010001 CLASS(SERVAUTH) ID(JOE) ACCESS(READ)
SETROPTS RACLIST(SERVAUTH) REFRESH
```

Setting up access to BCPii REXX execs

This topic describes how to set up access to run BCPii REXX execs in the System REXX environment and in the TSO/E REXX environment.

Setting up access to the HWIREXX helper program

BCPii provides the HWIREXX helper application to allow easier execution of a BCPii REXX exec in the System REXX environment. (See "Executing a BCPii REXX exec in the System REXX environment" on page 273 for more information.)

In order to run a BCPii System REXX exec using the HWIREXX helper program, you must have at least READ authority to the HWI.HWIREXX.*execname* resource in the FACILITY class, where *execname* specifies a 1- to 8-character name of the System REXX exec that you want to be executed by the HWIREXX helper program. BCPii also requires that the FACILITY class be RACLIST-specified. The following example shows the RACF commands to accomplish this:

```
RDEFINE FACILITY HWI.HWIREXX.execname UACC(NONE)
PERMIT HWI.HWIREXX.execname CLASS(FACILITY) ID(userid) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Setting up an environment to run BCPii TSO/E REXX execs

In order to allow REXX execs running under TSO/E to use BCPii, BCPii must specifically be granted the authority to run under TSO/E and the proper host command environment must exist.

Setting up access for BCPii TSO/E REXX Execs: The TSO/E environment is an unauthorized program environment. BCPii normally requires its APIs to be invoked from a program-authorized application. An installation may choose to allow BCPii APIs to be run under TSO/E REXX by making a configuration update to the "TSO/E Commands and Programs" parmlib member (IKJTSOxx). The program HWIC1TRX must be added to the list of APF-authorized programs that may be called through the TSO Service Facility (AUTHTSF).

The following example shows the syntax required to add BCPii to this list:

```
AUTHTSF NAMES(HWIC1TRX)
```

To activate this change on a live system, issue the SET command: SET IKJTSO=xx; where *xx* is the two-character suffix of the IKJTSOxx parmlib member where the update was made.

Once this change is activated, the TSO/E user still requires SAF authorization to the correct BCPii profiles in order to successfully perform the desired BCPii operations.

Host command environment considerations: When setting up the environment for REXX execs using BCPii running under TSO/E or ISPF, be sure to check that you are not using a customized copy of the IBM-supplied host command environment modules, IRXTSPRM or IRXISPRM. If your installation is using its own copies of these modules, review and update as appropriate to add the following statements to ensure that the BCPii host command environment can be found.

```
SUBCOMTB_NEXT_SYSCALL DS 0C
SUBCOMTB_ENTRY_BCPPII EQU *
SUBCOMTB_NAME_BCPPII DC CL8'BCPII ' /* Name is BCPPII */
SUBCOMTB_ROUTINE_BCPPII DC CL8'HWIM1RTI' /* Routine is HWIM1RTI */
SUBCOMTB_TOKEN_BCPPII DC CL16' '
SUBCOMTB_NEXT_BCPPII DS 0C
```

| **Note:** Modules can be in LPALIB or any data set loaded as part of the LPA
| extension (LPALSTxx, fixed LPA (IEAFIXxx), MLPA (IEALPAxx), dynamic LPA via
| the SETPROG LPA,ADD command.)

BCPii startup and shutdown

The BCPii address space normally does not need to be started or shut down. BCPii initialization occurs during system IPL. If the configuration is correct, no further action is required. The address space remains active and ready to handle BCPii requests.

BCPii address space does not start up at IPL

If the HWIBCPii address space is not active after an IPL has been done, look for HWI* messages in the system log. Most of the time, these messages pinpoint the reason for the failure of BCPii to become active.

In most cases, the address space did not start for one of two main reasons:

1. The support element that controls the CPC that contains the image of z/OS on which BCPii is being started has the improper configuration. Make sure all the steps have been followed in “Setting up connectivity to the support element” on page 258.
2. The community name of the local CPC is either not defined in the security product or contains an incorrect value. This is accompanied by message HWI022I (when the value defined in the security product is incorrect). See “Community name defined in the security product for each CPC” on page 263 for detailed information.

When these problems have been corrected, restart the BCPii address space. See “Restarting the HWIBCPii address space” for more information.

Ending the HWIBCPii address space

The application of certain kinds of code maintenance or other unusual circumstances might require that the BCPii address space be stopped. To stop the BCPii address space, issue the STOP command for the BCPii address space: P HWIBCPii. In most cases, the address space ends normally. BCPii services are no longer available until the address space is restarted. See *z/OS MVS Initialization and Tuning Reference* for more information about the STOP HWIBCPii command.

If the STOP command fails to completely bring down the BCPii address space, you can issue the CANCEL command: C HWIBCPii. The address space then ends in a similar way to the STOP command. See *z/OS MVS Initialization and Tuning Reference* for more information about the CANCEL command.

If the CANCEL command still fails to completely bring down the BCPii, you can issue the FORCE command as a last resort: FORCE HWIBCPii. See *z/OS MVS Initialization and Tuning Reference* for more information about the FORCE command.

BCPii issues an ENF 68 broadcast to notify interested ENF listeners that BCPii services are no longer available. See *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for more information regarding this ENF signal.

Restarting the HWIBCPii address space

After the BCPii address space has ended, it can be restarted. A procedure supplied by IBM in SYS1.PROCLIB allows the BCPii address space to be restarted. Issue the S HWISTART command to restart the HWIBCPii address space. When message HWI001I appears, BCPii is now active and all BCPii requests may resume. However, all prior connections are no longer valid, and applications will need to

re-establish these connections in order to resume their current BCPii activity. See *z/OS MVS Initialization and Tuning Reference* for more information about the START HWISTART command.

BCPii issues an ENF 68 broadcast when the address space has completely initialized to notify interested ENF listeners that BCPii services are now available. See *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for more information regarding this ENF signal.

SMF recording in BCPii

BCPii automatically writes SMF records for all HWISET and HWICMD requests that complete with a return code of zero.

In order for SMF to keep these records, it is necessary to activate recording of SMF record type 106 (X'6A') on the system. You can do this in either of the following ways:

- Create or modify an SMFPRM $_{xx}$ parmlib member and specify SYS(TYPE(106)) to capture all type 106 records, or SYS(TYPE(106(1))) to only capture those records written by HWISET or SYS(TYPE(106(2))) to only capture those records written by HWICMD. Then, activate the parmlib member by issuing the **SET SMF=xx** command.
- Dynamically add recording of type 106 records by issuing the **SETSMF** command.

Once activated, HWISET calls or HWICMD calls or both that complete with a return code of zero will be recorded in SMF.

When you want to review the records, you must dump them using either the SMF data set dump program (IFASMFDP) or the SMF logstream dump program (IFASMFDL), depending on whether your installation uses data set recording or logstream recording for SMF records. You can find a sample invocation of each of these programs in the first step of the IBM-supplied HWI6AFMT job in SYS1.SAMPLIB.

After dumping the records, you can analyze them using either of the following methods:

- **Use the IBM-supplied HWI6AFMT sample reporting job**

IBM provides a sample report using the DFSORT ICETOOL. By using simple JCL, you can create reports of the BCPii SMF data. The HWI6AFMT job requires the companion HWIRPTMP sample job to make the ICETOOL program aware of structure and formatting for the BCPii SMF data. You can copy and customize the HWI6AFMT job to report on the desired data.

- **Sort and format the records manually**

You might choose to write your own program to report on the type 106 records. The records are mapped by the HWISMF6A mapping macro found in SYS1.MACLIB.

BCPii callable services

You can use base control program internal interface (BCPii) services to connect an authorized z/OS application to System z configuration resources (such as CPC, image, capacity record, or activation profile data) and to allow that application to potentially modify these resources.

To use base control program internal interface (BCPii) services, issue calls from high level language programs. Each service requires a set of parameters coded in a specific order on the CALL statement.

This topic describes the CALL statements that invoke BCPii services. Each description includes a syntax diagram, parameter descriptions, return and reason code explanations with recommended actions. Return and reason codes are shown in hexadecimal and decimal with the associated equate symbols.

This topic contains the following subtopics:

- “Syntax, linkage and programming considerations”
- “HWICMD — Issue a BCPii hardware management command” on page 282
- “HWICONN — Establish a BCPii connection” on page 300
- “HWIDISC — Release a BCPii connection” on page 311
- “HWIEVENT — Register or unregister for BCPii events” on page 317
- “HWILIST — Retrieve HMC and BCPii configuration-related information” on page 329
- “HWIQUERY — BCPii retrieval of SE/HMC-managed attributes” on page 341
- “HWISET — BCPii set SE/HMC-managed attributes” on page 370
- “HWIBeginEventDelivery — Begin delivery of BCPii event notifications” on page 399
- “HWIEndEventDelivery — End delivery of BCPii event notifications” on page 403
- “HWIManageEvents — Manage the list of BCPii events” on page 406
- “HWIGetEvent — Retrieve outstanding BCPii event notifications” on page 410

Syntax, linkage and programming considerations

Programming language definitions are provided in the following languages:

- In C (HWICIC) in data set SYS1.SIEAHDRV.H. Miscellaneous C constants are defined in HWIZHAPI in the same data set.
- In REXX (HWICIREX) in data set SYS1.MACLIB. Miscellaneous REXX constants are defined in HWIC2REX in the same data set.

Note:

1. If the REXX exec is running under System REXX using the TSO=YES environment, these include files may be read in at the time of execution by the REXX exec. A simple programming example that reads the values into the REXX exec through the use of the EXECIO function is provided in the IBM-supplied REXX samples. See “Programming Examples” on page 281 for further information.
 2. If the REXX exec is running under System REXX using the TSO=NO environment, the definitions in these include files may be copied into the REXX exec.
- In assembler (HWICIASM) in data set SYS1.MACLIB. Miscellaneous assembler constants are defined in HWIC2ASM in the same data set.

Calling formats

Some specific calling formats for languages that can invoke the BCPii callable services are:

C BCPii_service_name (return_code,param1,param2, ...)

```
REXX ADDRESS BCPii "BCPii_service_name return_code parm1 parm2 ..."
```

Assembler Call macro

```
CALL BCPii_service_name,(return_code,parm1,parm2, ...),VLIST
```

BCPii connection scope

BCPii limits access to active BCPii connections. BCPii will not allow a program to use a previously established BCPii connection unless it is running in the proper environment. BCPii associates a connection with either an address space or a task, depending on the execution environment of the connector. It then uses this association (affinity) to determine if the connection is allowed to be used on subsequent requests.

Connections with address space affinity

The BCPii connections created by a C program, an assembler program, or a System REXX exec are associated with an address space.

- For C and assembler programs, BCPii creates an affinity between the connection and the address space that initiated the connection (via the HWICONN service).
- For a System REXX exec, BCPii creates an affinity between the connection and the address space that initiated the execution of the REXX exec (via the AXREXX authorized service call).

BCPii allows any task running in the same address space to use these connections on subsequent BCPii API calls. In addition, the connection remains active until the address space terminates.

Connections with task affinity

The BCPii connections created by a REXX exec running in either a TSO/E or ISV-provided REXX environment are associated with the task that initiated the execution of the REXX exec.

BCPii only allows the task that initiated the connection (via the HWICONN service) to access this connection on subsequent BCPii API calls. In addition, the connection only remains active until the task terminates.

Linkage considerations

There are two ways for a compiled BCPii application (non-REXX) to find BCPii callable services:

- Use the linkable stub routine HWICSS from SYS1.CSSLIB to link-edit your object code.
- Use the LOAD macro to find the address of the BCPii callable service at run time and then CALL the service.

REXX programming considerations

BCPii supports REXX execs being executed from the System REXX, TSO/E REXX, and independent software vendor (ISV) REXX programming environments. Each REXX environment is unique:

- System REXX supports all BCPii APIs and provides the capability to write sophisticated BCPii applications by utilizing REXX and other programming languages as part of a single application.

Note:

- To use the HWIEVENT and HWICMD services, a non-REXX adjunct helper program is needed to call z/OS system services to prepare for events and to coordinate with an event exit. See “Programming Examples” on page 281 for detailed information.
- The System REXX "MODIFY AXR" command is not supported by BCPii. See “Executing a BCPii REXX exec in the System REXX environment.”
- TSO/E REXX execs are easy to execute from a TSO user. This environment supports all the BCPii APIs, except HWIEVENT and HWICMD.
- ISV-provided REXX environments provide different features, depending on which ISV product is being used. These environments support all the BCPii APIs, except HWIEVENT and HWICMD.

The following table identifies the z/OS BCPii APIs supported in the three REXX environments:

Table 55. BCPii APIs supported in the REXX environment

BCPii APIs	System REXX environment	TSO/E REXX environment	ISV-provided REXX environment
HWICONN	X	X	X
HWIDISC	X	X	X
HWILIST	X	X	X
HWIQUERY	X	X	X
HWISET	X	X	X
HWIEVENT	X		
HWICMD	X		

The syntax of the BCPii REXX execs are identical in all three REXX environments. Therefore, a BCPii REXX exec written to be used in one REXX environment can be run in another REXX environment without change.

Executing a BCPii REXX exec in the System REXX environment

BCPii supports the invocation of its APIs from the System REXX programming environment. Execs running in this environment are APF-authorized. A user may choose either of the following methods to have their exec run under System REXX:

- Invoke the authorized HWIREXX helper program for basic requests.
- Use the AXREXX macro from an authorized program for more customized requests.

The dataset where the REXX exec is to be run must be specified using the REXXLIB keyword in the AXRxx parmlib member, and users of this program must have the proper authority to run programs residing in LINKLIB.

BCPii REXX programming restrictions for the System REXX environment: BCPii does not support being invoked from a REXX exec which has been started via the MODIFY AXR command. Any attempt to run from this environment results in a return code of HWI_REXXInvalidExecutionEnv.

Using the HWIREXX interface: For basic REXX execs, BCPii API calls can be run easily from the System REXX programming environment using the supplied HWIREXX helper program, without the need to code an assembler program with an AXREXX macro invocation. IBM provides sample invocation JCL for HWIREXX in SAMPLIB member HWIXMRJL.

The HWIREXX interface provides some of the most common AXREXX macro keywords as input parameters. Table 56 lists the supported keywords.

Table 56. HWIREXX keywords

HWIREXX keyword	Required/ Optional	Default value	AXREXX macro parameter equivalent
NAME=xxx; where xxx is a 1-8 character exec name to be executed.	Required	N/A	NAME
DSN=xxx.xxx.xxx; where xxx.xxx.xxx is a 1-44 character PDS data set name where the REXX exec output is directed. Note: The data set may be pre-allocated prior to execution of the exec. If the data set is not pre-allocated, the data set is allocated by System REXX. In either case, the output from the REXX exec is contained in a member name within the data set that matches the specified HWIREXX NAME.	Optional	NO_REXXOUTDSN	REXXOUTDSN
TSO=<Y/N>; where 'Y' means to run in the TSO host command environment, and 'N' means to run in the standard MVS host environment.	Optional	N	TSO
SYNC=<Y/N>; where 'Y' means the request is synchronous, and 'N' means the request is asynchronous.	Optional	Y	SYNC
TIMELIM=<Y/N>; where 'Y' means that a time limit is applied, and 'N' means that no time limit is applied.	Optional	Y	TIMELIMIT
TIME=xxx; where xxx is a number value between 1 and 21474536 that represents the number of seconds to allow the exec to run.	Optional	System default value	TIMEINT

See the JCL example HWIXMRJL shipped in SAMPLIB for more information on the invocation of the HWIREXX helper program.

If additional AXREXX macro parameters are required (other than the AXREXX macro parameters listed in Table 56) to properly establish the System REXX environment, an explicit invocation of the AXREXX macro is required. See "Using the AXREXX macro" on page 276 for detailed information.

Return codes from the HWIREXX service:

Table 57. Return codes from the HWIREXX service

HWIREXX return code (in decimal)	Meaning and action
0	Meaning: BCPii processed the REXX host command successfully. Action: Consult the BCPii return code on the BCPii service call to determine the final result of the request.
100	Meaning: Program error. Caller's JCL string has a syntax error. Action: Check for a probable coding error and correct the problem. See "Using the HWIREXX interface" on page 273 for detailed information.

Table 57. Return codes from the HWIREXX service (continued)

HWIREXX return code (in decimal)	Meaning and action
101	<p>Meaning: Program error. A required parameter is not found.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
102	<p>Meaning: Program error. No input parameters were specified.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
103	<p>Meaning: Program error. A parameter keyword was provided that is not supported by HWIREXX.</p> <p>Action: Check for a probable coding error and correct the problem. HWIREXX supports these keywords only: NAME, DSN, TSO, SYNC, TIMELIM, and TIME (which correspond to the AXREXX macro parameters: NAME, REXXOUTDSN, TSO, SYNC, TIMELIMIT, and TIMEINT, respectively.)</p>
104	<p>Meaning: Program error. Duplicate parameter keys are specified.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
105	<p>Meaning: Program error. A keyword may only consist of alphanumeric characters.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
106	<p>Meaning: Program error. Parameter values may only consist of alphanumeric characters and periods (.) .</p> <p>Action: Check for a probable coding error and correct the problem.</p>
107	<p>Meaning: Program error. The TSO parameter must be Y or N.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
108	<p>Meaning: Program error. The SYNC parameter must be Y or N.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
109	<p>Meaning: Program error. The TIMELIM parameter must be Y or N.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
110	<p>Meaning: Program error. A parameter value is too long. Name values are limited to 8 characters; data set names are limited to forty-four (44) characters; the TSO value is one character; the SYNC value is one character; the TIMELIM value is one character; and the TIME value is limited to 8 characters.</p> <p>Action: Check for a probable coding error. Reduce the length to the appropriate size based on the specified parameter.</p>
111	<p>Meaning: Program error. Blank character is not allowed in the JCL string.</p> <p>Action: Check for a probable coding error and correct the problem.</p>
112	<p>Meaning: Setup error. The caller does not have the correct SAF authorization to run the HWIREXX program.</p> <p>Action: The security administrator needs to give the user at least READ authority to the HWI.HWIREXX.execname resource in the FACILITY class. See "Setting up access to the HWIREXX helper program" on page 268 for more information.</p>

|
|
|
|
|
|

Table 57. Return codes from the HWIREXX service (continued)

HWIREXX return code (in decimal)	Meaning and action
2049 - 4111	<p>Meaning: Reason code returned from AXREXX.</p> <p>Action: See the AXREXX macro in <i>z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN</i>.</p>
4095	<p>Meaning: System error. An unexpected error is detected. The system rejects the service call.</p> <p>Action: Search the problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Using the AXREXX macro: If HWIREXX does not provide the options for your REXX exec requires, you can run your REXX exec using the AXREXX macro from the System REXX programming environment.

For example, an assembler program running in supervisor state, PKM 0-7, or APF-authorized can invoke the AXREXX macro to execute a REXX exec as follows:

```
AXREXX REQUEST=EXECUTE,
      NAME=execname,      <--- 8-character name of REXX exec
      TSO=NO,             <--- Runs in a standard MVS host command environment
      REXXARGS=rexxargs, <--- Input/output parmeters mapped by AXRARGLST
      REXXOUTDSN=outdsn, <--- Specify output data set
      REXXOUTMEMNAME=memname, <--- Specify output member name
      RETCODE=retcode,    <--- R15 as a result of REXX exec
      RSNCODE=rsncode,    <--- R10 as a result of REXX exec
      TIMELIMIT=[YES,NO], <--- Do you want the REXX exec to timeout?
      TIMEINT=numofsecs   <--- If TIMELIMIT=YES, how much time to wait?
```

After the invocation of the AXREXX macro, the REXX exec gets control and the input parameters are passed to the REXX exec. If any output is generated from the exec, it is directed to the specified output data set and member name. Lastly, the return code and reason code are returned.

For a complete description of the AXREXX macro and its usage, see *z/OS MVS Programming: Authorized Assembler Services Guide* and *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*. For a BCPii example showing the invocation of the AXREXX macro, see SAMPLIB member HWIXMRA1.

Executing a BCPii REXX exec in the TSO/E REXX environment

BCPii supports the invocation of its APIs from the TSO/E REXX programming environment, as long as the installation has allowed BCPii to be available from the TSO/E environment. See "Setting up an environment to run BCPii TSO/E REXX execs" on page 268 for information on setting up BCPii to run in a TSO/E REXX environment.

BCPii APIs can be run from REXX execs under TSO/E in the following ways:

- TSO/E foreground:
 - Issue the exec from the TSO/E READY mode, or
 - ISPF by using the TSO EXECUTE command.

See *TSO/E REXX User's Guide* for the syntax of the EXECUTE command.
- TSO/E background:

- Issue the exec from JCL, specifying IKJEFT01 as the program name on the JCL EXEC statement. See *TSO/E REXX Reference* for more information about running REXX execs using IKJEFT01.

BCPii REXX programming restrictions for the TSO/E environment: The following are not supported in BCPii REXX execs running in the TSO/E environment:

- HWICMD
- HWIEVENT
- HWI_LIST_EVENTS for the BCPii HWILIST service

Executing a BCPii REXX exec in an ISV-provided REXX environment

BCPii supports the invocation of its APIs from ISV-provided REXX programming environments, provided that the REXX execs running in this environment are program-authorized.

Because BCPii support is not native to ISV-provided REXX environments, the BCPii host command environment must first be enabled. To accomplish this, the BCPii REXX exec must first invoke the BCPii-provided *hwihost* function to enable the BCPii host command environment prior to any BCPii API invocation using "*address bcpii*".

Note: It is also recommended (but not required) that you invoke the *hwihost* function to disable the BCPii host environment when it is no longer needed by the BCPii REXX exec.

To enable the BCPii host command environment, add the following statement to your BCPii REXX exec:

```
RC = hwihost("ON")
```

To disable the BCPii host command environment, add the following statement to your BCPii REXX exec:

```
RC = hwihost("OFF")
```

Invocations of the *hwihost* function in an exec running in either the System REXX or TSO/E REXX programming environments are ignored, and the resulting return code is always zero. This ensures compatibility of REXX execs running in any REXX programming environment on z/OS.

BCPii REXX programming restrictions for an ISV-provided REXX environment: The following are not supported in BCPii REXX execs running in an ISV-provided REXX environment:

- HWICMD
- HWIEVENT
- HWI_LIST_EVENTS for the BCPii HWILIST service

REXX Programming tips

When programming a BCPii application using REXX, see the specific REXX programming considerations for each individual BCPii callable service for all necessary interface distinctions. Users of the BCPii REXX interface should be aware of the following:

- All parameters passed on BCPii REXX service calls must be REXX variables. Literals are not supported (for example, a variable name which has been assigned the value of a ListType should be specified on the call instead of the value itself).
- Variable names specified on BCPii REXX service calls are limited to 40 characters in length.
- Output variables specified on BCPii REXX service calls may be initialized or un-initialized. On input, the value of output variables are not verified. Output variables are initialized and set by BCPii.
- If the value of an input variable is incompatible with the parameter type required on a particular BCPii REXX service call, an error is flagged. See the REXX programming considerations for each BCPii callable service for the specific interface distinctions.
- The DiagArea for each BCPii REXX service call is returned using stem variables in the form: *x*.Diag_Index, *x*.Diag_Key, *x*.Diag_Actual, *x*.Diag_Expected, *x*.Diag_CommErr and *x*.Diag_Text (where *x* is the name of the stem variable specified on the parameter list). If no DiagArea information is filled in by BCPii, the value of the DiagArea stem-variable on return is all blanks.
- Stem variables utilized by BCPii have hard-coded stem variable tail values which usually correspond to the documented parameter name. For example, the QueryParm. stem must be prepared in REXX with the exact stem variable "ATTRIBUTEIDENTIFIER".
- The ConnectToken parameter returned on the HWICONN call and passed as input on all subsequent services contains non-displayable characters. Ensure that this ConnectToken is untouched by the REXX exec, thereby allowing subsequent BCPii services to read the value correctly.
- For System REXX execs only: Consider the length of time necessary to run your BCPii REXX exec. BCPii applications are interacting with the CPC's support element. Therefore, BCPii REXX execs may take longer to run than other REXX execs. To avoid having your BCPii REXX application end prematurely, even when the amount of time calculated is reasonable to complete your BCPii REXX exec, consider using the TIMELIMIT and TIMEINT keywords on the AXREXX service call. The default TIMELIMIT=YES, TIMEINT=SYSTEM causes the REXX exec to stop running after a predetermined amount of time. The TIMEINT value may be increased to give the REXX exec additional time to complete its execution before being timed out by the system. In certain circumstances, it may be necessary to specify TIMELIMIT=NO to prevent the REXX exec from timing out. This option should be used with caution as System REXX has a finite number of system-wide regions where the System REXX execs are executed. If TIMELIMIT=NO is specified unnecessarily, this could eventually lead to a constrained System REXX environment.
- BCPii connections created under System REXX can be used by any program running in the address space of the connector (Address space affinity). BCPii connections created under the TSO/E or ISV-provided REXX environments can only be used by the same task as the connector (Task affinity). See "BCPii connection scope" on page 272 for detailed information.
- BCPii requires all callers to be program-authorized. REXX execs in the zFS cannot run as APF-authorized when invoked from the shell. Therefore, any calls to BCPii services from REXX execs in this environment will result in a HWI_AUTH_FAILURE return code.
- The built-in REXX RC variable contains the return code from the REXX BCPii host command. This return code indicates BCPii's acceptance of the supplied REXX BCPii host command. The return codes returned in the RC variable are

generally unique to the REXX environment. In contrast, the BCPii service return code, the variable supplied on the service call itself, is only filled in if the RC variable has a value of HWI_OK (0) or HWI_REXXParmSyntaxError (1). Possible return codes returned by BCPii in the RC variable are:

Return codes from a REXX BCPii host command

Table 58. Return codes from a REXX BCPii host command

REXX RC returned from a BCPii host command (in decimal)	Meaning and action
0 HWI_OK	<p>Meaning: BCPii processed the REXX host command successfully.</p> <p>Action: Consult the BCPii return code on the BCPii service call to determine the final result of the request.</p>
1 HWI_REXXParmSyntaxError	<p>Meaning: Program error. The REXX BCPii host command has detected that the format of the parameters is not in the proper form to be accepted by BCPii.</p> <p>Action: Check for a probable coding error. See the BCPii return code on the BCPii service call to determine the reason for the syntax error. See the REXX programming considerations of the BCPii service to see the exact calling specifications. Compare the BCPii REXX service call attempted with service call examples in the supplied BCPii REXX programming sample found in SYS1.SAMPLIB. See the DiagArea for further diagnostic information.</p>
2 HWI_REXXUnsupportedService	<p>Meaning: Program error. An unknown BCPii service name was specified on the BCPii REXX host command.</p> <p>Action: Check for a probable coding error. Specify a valid BCPii service name (for example, HWICONN, HWILIST, and so on).</p>
3 HWI_REXXInvalidNumofParms	<p>Meaning: Program error. The number of parameters specified on the BCPii REXX host command for the service name specified does not match the number of parameters expected.</p> <p>Action: Check for a probable coding error. See the REXX programming considerations of the BCPii service to see the exact calling specifications. Compare the BCPii REXX service call attempted with service call examples found in the supplied BCPii REXX programming sample found in SYS1.SAMPLIB.</p>
4 HWI_REXXStemVarRequired	<p>Meaning: Program error. The BCPii REXX service specified on the BCPii REXX host command is missing one or more required stem variables in the positional parameter list.</p> <p>Action: Check for a probable coding error. See the REXX programming considerations of the BCPii service to see the exact calling specifications. A stem variable parameter must specify a "." following the variable name (for example, "var."). Also, compare the BCPii REXX service call attempted with service call examples found in the supplied BCPii REXX programming sample found in SYS1.SAMPLIB.</p>
5 HWI_REXXParmNameTooLong	<p>Meaning: Program error. One or more variables specified on the BCPii REXX service call on the BCPii REXX host command is greater than the BCPii maximum REXX variable length (40).</p> <p>Action: Check for a probable coding error. Reduce the variable name lengths on the BCPii REXX service call to be 40 characters or less in length.</p>

Table 58. Return codes from a REXX BCPii host command (continued)

REXX RC returned from a BCPii host command (in decimal)	Meaning and action
6 HWI_REXXInvalidHostEnv	<p>Meaning: System error. BCPii detected an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
7 HWI_REXXInvokerNotFound	<p>Meaning: Program error. The address space issuing the AXREXX invocation is no longer running. No new BCPii connections are allowed.</p> <p>Action: Determine the reason that the AXREXX-invoking address space terminated prior to the termination of the REXX exec. Correct the situation and start again.</p>
8 HWI_REXXInvalidExecutionEnv	<p>Meaning: Program error. BCPii does not support the BCPii host command running in the current execution environment.</p> <p>If the current execution environment is System REXX, it may mean that an attempt was made to issue a BCPii host command from an exec that was started using the MODIFY AXR command.</p> <p>If the current execution environment is either TSO/E or ISV-provided REXX, it may mean that the requested service was not supported in this environment.</p> <p>Action: Run the BCPii host command from a supported environment.</p>
9 HWI_REXXUnsupportedListType	<p>Meaning: Program error. BCPii does not support the specified ListType on the BCPii HWILIST service in the current execution environment.</p> <p>Action: Correct the specified ListType value or try this request again in a valid execution environment (for example, the System REXX environment).</p>
32 HWI_REXXInternalSystemError	<p>Meaning: System error. BCPii detected an unexpected error while invoking REXX services. The system rejects the service call.</p> <p>Action: A symptom record has been written to LOGREC to record the problem. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
4095 HWI_Unexpected_Error	<p>Meaning: System error. BCPii detected an unexpected error. The system rejects the service call.</p> <p>Action: A symptom record has been written to LOGREC to record the problem. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

REXX return codes from the BCPii *hwihost* function

Table 59. REXX return codes from the BCPii *hwihost* function. The following return codes apply only to callers running their BCPii REXX execs in an ISV-provided REXX environment.

REXX RC returned by the BCPii <i>hwihost</i> function	Meaning and action
1 HWI_hwihost_ParmSyntaxError	<p>Meaning: Program Error. The specified argument is not "ON" or "OFF".</p> <p>Action: Check for a probable coding error. Try this request again with an argument of "ON" or "OFF".</p>

Table 59. REXX return codes from the BCPii hwihost function (continued). The following return codes apply only to callers running their BCPii REXX execs in an ISV-provided REXX environment.

REXX RC returned by the BCPii hwihost function	Meaning and action
2 HWI_hwihost_InternalSystemError	<p>Meaning: System error. BCPii detected an unexpected error while invoking TSO/E REXX services. The system rejects the service call.</p> <p>Action: A symptom record has been written to LOGREC to record the problem. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Sample REXX exec

Here is a sample REXX exec using BCPii calls that lists the names of all of the interconnected CPCs and then attempts to connect to each one of them:

```

/* REXX */
ListType = HWI_LIST_CPCS;
Address BCPii "HWILIST Retcode ConnectToken ListType AnswerArea.
DiagArea."

If RC = 0 & retcode = 0 Then
  Do
    ConnectType = HWI_CPC
    Do i = 1 To AnswerArea.0
      Say "CPC" i ":" AnswerArea.i

      InConnectToken = 0
      Address BCPii "HWICONN Retcode InConnectToken OutConnectToken
ConnectType AnswerArea.i DiagArea."
      If RC = 0 & retcode = 0 Then
        Say "Connected to CPC "AnswerArea.i"."
    End
  End

```

For REXX execs running in an ISV-provided environment, make sure to add the following line prior to the first address BCPii statement:

```
RC = hwihost("ON")
```

Assembler programming considerations

Callers must also use the following linkage conventions:

- Register 1 must contain the address of a parameter list that is a list of consecutive words, each containing the address of a parameter to be passed. The last word in this list must have a 1 in the high-order (sign) bit.
- Register 13 must contain the address of an 18-word save area.
- Register 14 must contain the return address.
- Register 15 must contain the entry point address of the service being called.
- If the caller is running in AR ASC mode, access registers 1, 13, 14, and 15 must all be set to zero.

On return from the service, general and access registers 2 through 14 are restored (registers 0, 1 and 15 are not restored).

Programming Examples

BCPii provides sample programs to aid in the creation of BCPii applications in both C and REXX programming languages. The samples are shipped in SYS1.SAMPLIB.

HWIXMCS1 (Metal C programming language) provides an example of how to use all of the BCPii APIs and how to construct a simple BCPii application. HWIXMCX1 (Metal C programming language) provides a simple example of how a BCPii Event Notification Facility (ENF) exit could be coded to field various BCPii-registered events.

HWIXMRS1 (REXX programming language) provides an example of how to use the most common BCPii APIs. It can easily be invoked in the System REXX environment by utilizing the IBM-provided HWIREXX program using the provided sample JCL HWIXMRJL.

Another REXX sample (HWIXMRS2) is provided to show how a REXX application can utilize the HWIEVENT and HWICMD APIs. It is invoked using an AXREXX macro invocation in the sample assembler "helper" program (HWIXMRA1). This second sample can utilize the Metal C ENF exit HWIXMCX1.

HWICMD — Issue a BCPii hardware management command

Call the HWICMD service to perform a command against an HMC-managed object that is associated with central processor complexes (CPCs) and CPC images (LPARs). User-defined image groups can also be utilized to target multiple images with a single command.

BCPii commands, because of the very nature of what they are attempting to do, may take a significant amount of time to complete. To prevent applications from being tied up for an excessive amount of time while waiting for the command to complete, HWICMD will return to the caller either when the command has been *accepted* by the target support element (SE) or when the command was found to contain errors. The actual completion of the command can be determined by consulting the final return code returned in the BCPii command response event.

To receive this BCPii command response event, an application must have registered for the Hwi_Event_CmdResp event prior to the HWICMD invocation. Registration for this or any event is accomplished by calling the HWIEVENT service, or for z/OS UNIX callers, by calling HwiManageEvents. The HWIEVENT service requires a user-supplied Event Notification Facility (ENF) exit.

When the command completes, BCPii will signal the ENF to notify registered applications that a command response has been received. For non-z/OS UNIX callers, the ENF exit specified will receive control and the command response event returned data will contain the final return code of the request. For z/OS UNIX callers, the HwiGetEvent service can be used to receive the event notification and to determine the final return code of the HWICMD service.

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	One of the following: PKM allowing key 0-7, supervisor state, or APF-authorized
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)

Requirement	Details
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space and addressable by the caller
Linkage:	Standard MVS linkage conventions are used

Programming requirements

See “Syntax, linkage and programming considerations” on page 271 for details about how to call BCPii services in the various programming languages.

The microcode level that supports the command service call (HWICMD) of BCPii is required to be installed on the target CPC. See the HWI_CMD_NOT_SUPPORT_WARNING return code in “HWICONN — Establish a BCPii connection” on page 300 for more information.

See “HWICMD” on page 596 for the summary table of the BCPii HWICMD types and the objects that can be targeted for each command.

REXX programming considerations for the HWICMD service

All information for the HWICMD service applies for REXX requests except:

- A stem variable (for example, CmdParm.) replaces CmdParm_ptr.
- The CmdParm structure names in Table 60 on page 288 are used as the dot-qualified names in the CmdParm stem variable. The following are exceptions:
 - On the HWI_CMD_POWER_CONTROL, HWI_CMD_TEMPCAP, and HWI_CMD_SYSPLEX_TIME_SET_STP_CONFIG commands, XML replaces XML_ptr and XML_Size is ignored.
 - On the HWI_CMD_SYSRESET_IPLT command, IPL-Token replaces IPL-Token_Ptr and IPL-Token_Len is ignored.

Restrictions

- BCPii does not allow any HWICMD to be targeted to a CPC that is earlier than a z9 platform.
- BCPii does not allow HWICMD to be issued from within a BCPii ENF exit routine.
- BCPii does not allow any HWICMD to be issued from a REXX exec running in a TSO/E or ISV-provided REXX environment.

Authorization

The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

The client application must have at least control access to the following SAF-protected FACILITY class resource profiles:

- HWI.TARGET.netid.nau for a ConnectToken that represents a CPC connection or an image group connection.
- HWI.TARGET.netid.nau.imagename for a ConnectToken that represents an image connection.
- HWI.TARGET.netid.nau.imagename for all individual images within the image group for a ConnectToken that represents a user-defined image group.

HWICMD

Note: BCPii requires the FACILITY class to be RACLIST-specified.

SMF recording

Requests that complete with a return code of zero will have SMF type 106 (X'6A') records written if the installation has activated recording of this record type in its active configuration.

Syntax

Write the call as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Non-REXX parameters	REXX parameters
CALL HWICMD(ReturnCode, ConnectToken, CmdType, CmdParm_Ptr, DiagArea);	address bcpil "hwicmd ReturnCode ConnectToken CmdType CmdParm. DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

ReturnCode contains the return code from the service.

ConnectToken

Supplied parameter

- Type: Character string
- Length: 16 bytes

ConnectToken specifies the connect token that this command is executed against. A ConnectToken represents a logical connection between the application and a CPC or an image, and is returned as an output parameter on the HWICONN service call.

A ConnectToken representing a user-defined image group may also be specified. In this case, the command will be executed on all members in the group, and not just on a single image.

The ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call.

CmdType

Supplied parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

CmdType specifies the type of the requested command.

See the following publications for more information about how the various commands operate, what inputs are required, and what outputs are expected:

- *System z Application Programming Interfaces* (SB10-7030-13)
- *System z10 and eServer zSeries Application Programming Interfaces* (SB10-7030-09)

- *System z9 and eServer zSeries Application Programming Interfaces (SB10-7030-08)*
- *zEnterprise System Support Element Operations Guide (SC28-6896-02)*
- *System z10 Support Element Operations Guide (SC28-6858-02)*
- *System z9 Support Element Operations Guide (SC28-6858-01)*

Constant in Hexadecimal (Decimal) Equate Symbol	Description
1 (1) HWI_CMD_ACTIVATE	Activate request to start target systems with the default activation profile name (HWI_APROF) associated with a CPC or an image. Note: The input connection token represents a <i>CPC connection</i> , an <i>image connection</i> , or an <i>image group connection</i> . This command cannot be issued specifying a connect token that represents either the local CPC or the local image.
2 (2) HWI_CMD_DEACTIVATE	Deactivate request to close down target systems. Note: The input connection token represents a <i>CPC connection</i> , an <i>image connection</i> , or an <i>image group connection</i> . This command cannot be issued specifying a connect token that represents either the local CPC or the local image.
3 (3) HWI_CMD_HWMSG	Hardware messages request. Note: The input connection token must only represent a <i>CPC connection</i> .
4 (4) HWI_CMD_CBU	Capacity backup CPC feature operation. Note: The input connection token must only represent a <i>CPC connection</i> .
5 (5) HWI_CMD_OOCOD	On/Off capacity on demand request. Note: The input connection token must only represent a <i>CPC connection</i> .
6 (6) HWI_CMD_PROFILE	Access CPC activation profiles. Note: The input connection token must only represent a <i>CPC connection</i> .
7 (7) HWI_CMD_RESERVE	Set exclusive CPC control. Note: The input connection token must only represent a <i>CPC connection</i> .
8 (8) HWI_CMD_SYSRESET	System reset request for target systems. See Cmdtype HWI_CMD_SYSRESET_IPLT for the latest version of the Sysreset command. Note: The input connection token must only represent an <i>image connection</i> or an <i>image group connection</i> . This command cannot be issued specifying a connect token that represents the local image.
9 (9) HWI_CMD_START	Start request for all CPs on target systems. Note: The input connection token must only represent an <i>image connection</i> or an <i>image group connection</i> . This command cannot be issued specifying a connect token that represents the local image.
A (10) HWI_CMD_STOP	Stop request for all CPs on target systems. Note: The input connection token must only represent an <i>image connection</i> or an <i>image group connection</i> . This command cannot be issued specifying a connect token that represents the local image.
B (11) HWI_CMD_PSWRESTART	Restart request for one CP on target system. The first CP that is found to be in the correct state is reset. Note: The input connection token must only represent an <i>image connection</i> or an <i>image group connection</i> . This command cannot be issued specifying a connect token that represents the local image.

HWICMD

Constant in Hexadecimal (Decimal) Equate Symbol	Description
C (12) HWI_CMD_OSCMD	Send operating system command request. Note: The input connection token must only represent an <i>image connection</i> .
D (13) HWI_CMD_LOAD	Load request to IPL target operating systems. Note: The input connection token must only represent an <i>image connection</i> or an <i>image group connection</i> . This command cannot be issued specifying a connect token that represents the local image.
E (14) HWI_CMD_TEMPCAP	Addition or removal of temporary capacity. Note: The input connection token must only represent a <i>CPC connection</i> .
F (15) HWI_CMD_SYSRESET_IPLT	System reset request for target systems with IPL token correlation. This is an enhanced version of HWI_CMD_SYSRESET. Note: The input connection token must only represent an <i>image connection</i> .
10 (16) HWI_CMD_ACTIVATE _WITH_ACTPROF	Activate request to start target systems using a supplied activation profile name. This is an enhanced version of the HWI_CMD_ACTIVATE command. Note: The input connection token must only represent a <i>CPC connection</i> or an <i>image connection</i> .
11 (17) HWI_CMD_POWER_CONTROL	Control the power usage characteristics. Note: The input connection token must only represent a <i>CPC connection</i> .
12 (18) HWI_CMD_SCSI_LOAD	SCSI Load from FCP (Fibre Channel Protocol for SCSI) attached SCSI (Small Computer System Interface) disks. Note: The input connection token must only represent an <i>image connection</i> or an <i>image group connection</i> .
13 (19) HWI_CMD_SCSI_DUMP	SCSI Dump to FCP (Fibre Channel Protocol for SCSI) attached SCSI (Small Computer System Interface) disks. Note: The input connection token must only represent an <i>image connection</i> .
14 (20) HWI_CMD_SYSPLEX_TIME _SWAP_CTS	In a configured STP-only coordinated timing network (CTN), one CPC has the role of current time server (CTS). If the CTN has both a preferred time server and a backup time server configured, either one can be the CTS. This command swaps the role of CTS from preferred time server to backup time server or vice versa. The target system must be the system that will become the CTS. Note: The input connection token must only represent a <i>CPC connection</i> .
15 (21) HWI_CMD_SYSPLEX_TIME _SET_STP_CONFIG	This command sets the configuration for an STP-only coordinated timing network (CTN). The target system must be the system that will become the current time server (CTS). Note: The input connection token must only represent a <i>CPC connection</i> .
16 (22) HWI_CMD_SYSPLEX_TIME _CHANGE_STP_ONLY_CTN	This command, sent to the defined CPC with the role of current time server (CTS) in an STP-only coordinated timing network (CTN), changes the STP_ID portion of the CTN ID for the entire STP-only CTN. Note: The input connection token must only represent a <i>CPC connection</i> .

Constant in Hexadecimal (Decimal) Equate Symbol	Description
17 (23) HWI_CMD_SYSPLEX_TIME_JOIN_STP_ONLY_CTN	This command allows a CPC to join an STP-only coordinated timing network (CTN). The target system cannot be the current time server. If the CPC is already participating in an STP-only CTN, it will be removed from that CTN and joined to the specified one. If the CPC has an ETR ID, it will be removed. Note: The input connection token must only represent a <i>CPC connection</i> . Attention: Use extreme caution when issuing this command. Joining the STP-only CTN may result in a disabled wait state for all images that are in a parallel sysplex on the target CPC.
18 (24) HWI_CMD_SYSPLEX_TIME_LEAVE_STP_ONLY_CTN	This command removes a CPC from an STP-only coordinated timing network (CTN). The target system cannot be the current time server. Note: The input connection token must only represent a <i>CPC connection</i> . Attention: Use extreme caution when issuing this command. Leaving the STP-only CTN may result in a disabled wait state for all images that are in a parallel sysplex on the target CPC.

CmdParm_Ptr (non-REXX)**CmdParm. (REXX)**

Supplied parameter

- Type: Pointer (non-REXX), stem variable (REXX)
- Length: 4 bytes (non-REXX)

Non-REXX:

CmdParm_Ptr specifies the address of the command parameter that contains a structure of the input parameters for the requested command.

Take the following action according to the different conditions:

- For all optional parameters, callers are required to initialize the parameters to zero for BCPii to interpret them as null parameters unless otherwise specified.
- For commands with one or more required parameters and also with one or more optional parameters, callers are required to initialize each optional parameters to zero if they require BCPii to take the default action for that parameter.
- For commands that have only optional parameters, callers can initialize the CmdParm_Ptr to zero if they require BCPii to take the default action for all parameters.
- For commands that have no parameters, the CmdParm_Ptr is ignored.
- All string type parameters are required to be padded with trailing blanks unless otherwise specified.
- For commands that target image groups, the parameters specified in the CmdParm must be appropriate for all the images in the image group.

REXX:

CmdParm stem contains compound (stem) variables which represent input parameters for the requested command. The tail names of the stem variable are constants which must match the parameter names in Table 60 on page 288.

For optional parameters that are not initialized, BCPii interprets them as null parameters.

HWICMD

Table 60. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
ACTIVATE	HWI_CMD_ACT_PARM	ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) Note: Only a ForceType of HWI_CMD_FORCE will result in a successful activation of the target CPC or image if the target CPC or image is already active.
DEACTIVATE	HWI_CMD_DEACT_PARM	ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE)
HWMSG	HWI_CMD_HWMSG_PARM	HWMSGType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means REFRESH (HWI_CMD_HWMSG_REFRESH) • 2 – means DELETE (HWI_CMD_HWMSG_DELETE)
		HWMSGTimestamp	A null-terminated character string, up to 32 characters long. Required only for HWMSGType = HWI_CMD_HWMSG_DELETE. The timestamp specified must be an exact match of a timestamp returned on a HWMSGType = HWI_CMD_HWMSG_REFRESH request. An example of a timestamp: '08-20-2010 11:01: 23:145'. To delete a message, first run an HWI_CMD_HWMSG_REFRESH request to obtain the full timestamp and then issue the HWI_CMD_HWMSG_DELETE request, specifying the timestamp.
CBU	HWI_CMD_CBU_PARM	CBUType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means ACTIVATE (HWI_CMD_ACT) • 2 – means UNDO (HWI_CMD_UNDO)
		ActivateType	A 4-byte integer (required only for CBUType = HWI_CMD_ACT): <ul style="list-style-type: none"> • 1 – means REAL CBU (HWI_CMD_REAL) • 2 – means TEST CBU (HWI_CMD_TEST)
OOCOD	HWI_CMD_OOCOD_PARM	OOCODType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means ACTIVATE (HWI_CMD_ACT) • 2 – means UNDO (HWI_CMD_UNDO)
		OrderNumber	Required for OOCODType = HWI_CMD_ACT

Table 60. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
PROFILE	HWI_CMD_PROFILE_PARM	ProfileType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means IMPORT (HWI_CMD_PROFILE_IMPORT) • 2 – means EXPORT (HWI_CMD_PROFILE_EXPORT)
		AreaNumber	A 2-byte integer area number is required and must be in the range of 1 to 4.
RESERVE	HWI_CMD_RESERVE_PARM	ReserveType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means ADD (HWI_CMD_RESERVE_ADD) • 2 – means DELETE (HWI_CMD_RESERVE_DELETE)
		ApplName	An 8-character application name (required) padded with trailing blanks.
SYSRESET	HWI_CMD_SYSRESET_PARM	ResetType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means NORMAL (HWI_CMD_RESET_NORMAL) • 2 – means CLEAR (HWI_CMD_RESET_CLEAR)
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) <p>Note: Only a ForceType of HWI_CMD_FORCE will result in a successful sysreset of the target CPC or image if the target CPC or image is already active.</p>
START	0	N/A	N/A
STOP	0	N/A	N/A
PSWRESTART	0	N/A	N/A
OSCMD	HWI_CMD_OSCMD_PARM	PriorityType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means Priority (HWI_CMD_PRIORITY) • 2 – means Non-Priority (HWI_CMD_NONPRIORITY) <p>Note: For WTOR replies targeting a z/OS image, a PriorityType of Non-Priority may need to be specified to allow z/OS to receive the reply command.</p>
		OSCMDString	A 126-null-terminated character operating system command string (required).
LOAD	HWI_CMD_LOAD_PARM	LoadAddr	A 4-character string consisting only of hexadecimal characters identifying the device address to be used when performing the load (optional).
		LoadParm	An 8-character string as determined by the operating system being loaded (optional).

HWICMD

Table 60. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
		ForceType	<p>A 4-byte integer (optional, the default is FORCE):</p> <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) <p>Note: Only a ForceType of HWI_CMD_FORCE will result in a successful load of the target CPC or image if the target CPC or image is already active.</p>
TEMPCAP	HWI_CMD_TEMPCAP_Parm	TEMPCAPType	<p>A 4-byte integer (required):</p> <ul style="list-style-type: none"> • 1 – means Add (HWI_CMD_TEMPCAP_ADD) • 2 – means Remove (HWI_CMD_TEMPCAP_REMOVE)
		XML_Ptr (non-REXX)	<p>A character string pointer that points to the address of the XML information that illustrates the markup used to perform activation of the temporary capacity (required).</p>
		XML (REXX)	<p>XML information that illustrates the markup used to perform activation of the temporary capacity (required).</p>
		XML_Size (non-REXX)	<p>A 4-byte integer (required).</p> <p>Length in bytes of the XML that the XML_Ptr points to.</p>
SYSRESET _IPLT	HWI_CMD_SYSRESET _IPLT_PARM	ResetType	<p>A 4-byte integer (required):</p> <ul style="list-style-type: none"> • 1 – means NORMAL (HWI_CMD_RESET_NORMAL) • 2 – means CLEAR (HWI_CMD_RESET_CLEAR)
		ForceType	<p>A 4-byte integer (optional, the default is FORCE):</p> <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) <p>Note: Only a ForceType of HWI_CMD_FORCE will result in a successful sysreset of the target CPC or image if the target CPC or image is already active.</p>
		IPL-Token_Ptr (non-REXX)	<p>A character string pointer that specifies the address of the IPL token used to correlate a SYSRESET with other outstanding HMC-related activities. This ensures that this SYSRESET is operating with the same IPL instance as when the IPL-Token was retrieved (required).</p>

Table 60. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
		IPL-Token (REXX)	IPL token used to correlate a SYSRESET with other outstanding HMC-related activities. This ensures that this SYSRESET is operating with the same IPL instance as when the IPL-Token was retrieved (required).
		IPL-Token_Len (non-REXX)	A 4-byte integer (required). Length in bytes of the IPL token to which the IPL-Token_Ptr points.
ACTIVATE_ WITH _ACTPROF	HWI_CMD_ACT_WITH_ ACTPROF_PARM	ActProfName	A 16-character activation profile name padded with trailing blanks (required).
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) Note: Only a ForceType of HWI_CMD_FORCE will result in a successful activation of the target CPC or image if the target CPC or image is already active.
POWER _CONTROL	HWI_CMD_POWER _CONTROL_PARM	XML_Ptr (non-REXX)	A character string pointer that points to the address of the XML fragment describing the power characteristics to be applied to the CPC specified by the connect token (required).
		XML (REXX)	XML fragment describing the power characteristics to be applied to the CPC specified by the connection token (required).
		XML_Size (non-REXX)	A 4-byte integer (required). Length in bytes of the XML that the XML_Ptr points to.
SCSI_LOAD	HWI_CMD_SCSI_LOAD_PARM	LoadAddr	A 4-character string (optional) consisting only of hexadecimal characters (0-9, A-F) identifying the device address to be used when performing the SCSI load. Defaults to value last used when previous SCSI Load was performed.
		LoadParm	An 8-character string (optional) as determined by the operating system being loaded. Defaults to value last used when previous SCSI Load was performed.
		WW_Portname	A 16-character string (optional) identifying the World Wide Port Name to be used when performing a SCSI Load. Defaults to value last used when previous SCSI Load was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).

HWICMD

Table 60. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
		LU_Num	A 16-character string (optional) identifying the logical unit number (LUN) to be used when performing the SCSI Load. Defaults to value last used when previous SCSI Load was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		Boot_Pgm_Selector	A 4-byte integer (optional) identifying the boot program selector to be used for the SCSI Load. Defaults to value last used when previous SCSI Load was performed.
		Opsys_Loadparm	A 256-character string (optional) representing the operating system-specific load parameters to be used for the SCSI Load. Defaults to value last used when previous SCSI Load was performed.
		Bootrec_Blkl_Addr	A 16-character string (optional) representing the boot record logical block address to be used for the SCSI Load. Defaults to value last used when previous SCSI Load was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) <p>Note: Only a ForceType of HWI_CMD_FORCE will result in a successful load of the target CPC or image if the target CPC or image is already active.</p>
SCSI_DUMP	HWI_CMD_SCSI_DUMP_PARM	LoadAddr	A 4-character string (optional) consisting only of hexadecimal characters (0-9, A-F) identifying the device address to be used when performing the SCSI Dump. Defaults to value last used when previous SCSI Dump was performed.
		LoadParm	An 8-character string (optional) used when performing the SCSI dump. Defaults to value last used when previous SCSI Dump was performed.
		WW_Portname	A 16-character string (optional) identifying the World Wide Port Name to be used when performing a SCSI Dump. Defaults to value last used when previous SCSI Dump was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).

Table 60. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
		LU_Num	A 16-character string (optional) identifying the logical unit number (LUN) to be used when performing the SCSI Dump. Defaults to value last used when previous SCSI Load was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		Boot_Pgm_Selector	A 4-byte integer (optional) identifying the boot program selector to be used for the SCSI Dump. Defaults to value last used when previous SCSI Load was performed.
		Opsys_Loadparm	A 256-character string (optional) representing the operating system-specific load parameters to be used for the SCSI Dump. Defaults to value last used when previous SCSI Dump was performed. Note: If less than 256 bytes, a null terminator signifies the end of the string.
		Bootrec_Blk_Addr	A 16-character string (optional) representing the boot record logical block address to be used for the SCSI Dump. Defaults to value last used when previous SCSI Dump was performed. The character string must be comprised of hexadecimal values only (0-9, A-F).
		ForceType	A 4-byte integer (optional, the default is FORCE): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE) <p>Currently, either ForceType value will cause the same result. The target image will be dumped in either case. IBM recommends that an application omit this parameter.</p>
SYSPLEX_TIME_SWAP_CTS	HWI_CMD_SYSPLXTIME_SWAP_CTS_PARM	STP_ID	An 8-character non-terminated string (required) representing the current STP identifier associated with this CPC.
SYSPLEX_TIME_SET_STP_CONFIG	HWI_CMD_SYSPLXTIME_SET_STP_CONFIG_PARM	STP_ID	An 8-character non-terminated string (required) representing the current STP identifier associated with this CPC.
		ForceType	A 4-byte integer (required): <ul style="list-style-type: none"> • 1 – means Force YES (HWI_CMD_FORCE) • 2 – means Force NO (HWI_CMD_NOFORCE)
		XML_Ptr (non-REXX)	A character string pointer (required) points to the address of the XML fragment describing the configuration for the STP-only CTN.
		XML (REXX)	XML fragment describing the configuration for the STP-only CTN. (required)

HWICMD

Table 60. Structure pointed to by CmdParm_Ptr (non-REXX); CmdParm stem variable (REXX) (continued)

CmdType : HWI_CMD_	CmdParm (non-REXX)	Parameters in Structure (non-REXX) / Tail name constant of the user-defined CmdParm stem (REXX)	Parameter Values
		XML_Size (non-REXX)	A 4-byte integer (required). Length in bytes of the XML that the XML_Ptr points to.
SYSplex_TIME _CHANGE_STP _ONLY_CTN	HWI_CMD_SYSPLXTIME_CHG _STPONLYCTN_PARM	STP_ID	An 8-character non-terminated string (required) representing the desired STP identifier for the CPC and all CPCs that are members of the same STP-only CTN.
SYSplex_TIME _JOIN_STP _ONLY_CTN	HWI_CMD_SYSPLXTIME_JOIN _STPONLYCTN_PARM	STP_ID	An 8-character non-terminated string (required) representing the current STP identifier for the CPC.
SYSplex_TIME _LEAVE_STP _ONLY_CTN	0	N/A	N/A

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the DiagArea might not be filled in, and the data returned in the area should be ignored.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value that is specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code that is returned from the console application API or the BCPii transport layer.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, "BCPii communication error reason codes," on page 583 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0001yyyy' because of one of the following reasons:

Table 61. Reasons for abend X'042', RC X'0001yyyy'

yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See *z/OS MVS System Codes* for additional information.

Return codes

When the service returns control to the caller, GPR 15 and the ReturnCode contain a hexadecimal return code.

0 HWI_OK	<p>Meaning: The command has been accepted by the support element. An SMF record has been written.</p> <p>Action: Determine the final command completion result by consulting the return code value found in the data returned by the command response event. This ENF event is signaled if the application has already registered to receive this event (HWIEVENT or HwiManageEvents service).</p>
100 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified connect token is not valid. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The connect token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken. • The connect token does not represent an active connection. The connection specified might have already been disconnected using the HWIDISC service call. • The connect token is not associated with the caller's address space. The ConnectToken specified is associated with a different address space than the caller of this service call. <p>Action: Check for probable coding error.</p>
101 HWI_COMMUNICATION_ERROR	<p>Meaning: A communication error is detected. The hardware management console application API (HWMCA) or the BCPii transport layer has returned with a failing return code.</p> <p>Action: See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPii transport layer.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, "BCPii communication error reason codes," on page 583.</p>

HWICMD

102 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field.</p>
103 HWI_CONNECT_TOKEN_INACTIVE	<p>Meaning: The specified connect token is no longer valid. The connection has been disconnected or it is in the progress of being disconnected.</p> <p>Action: Check for probable coding error. Verify that the specified connect token is still active. If connectivity to the targeted CPC connection no longer exists, all connections associated with that CPC will no longer have a connect token that can be used.</p>
602 HWI_CMDTYPE_INV	<p>Meaning: Program error. The requested CMDTYPE specified in the call is not valid. The system rejects the service call. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The CmdType specified is not in the acceptable value range of possible command types. The Diag_Text indicates this error with the text of 'Invalid Cmd'. • The CmdType specified applies only to CPC connections, but the ConnectToken specified represents an image connection. The Diag_Text indicate this error with the text of 'Mismatch'. • The CmdType specified applies only to image connections, but the ConnectToken specified represents a CPC connection. The Diag_Text indicates this error with the text of 'Mismatch'. • The CmdType specified applies only to image connections, but the ConnectToken specified represents an image group connection. The Diag_Text will indicate this error with the text of 'Mismatch'. <p>Action: Check for probable coding error. Verify that the specified CmdType is in the acceptable value range. See the CmdType parameter section to verify that the specified connect token is applied for the requested command. See the DiagArea for further diagnostic information.</p>
603 HWI_CMDPARAM_INV	<p>Meaning: Program error. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • Required parameters are missing. • One or more parameters specified are not valid. <p>Action: Check for probable coding error. See the DiagArea for additional diagnostic information. The Diag_Index specifies the value of the CmdType parameter. The Diag_Text specifies the name of the parameter in the CmdParm structure. Note that the name might be abbreviated because of the limited size of the Diag_Text field.</p>

<p>604 HWI_CMD_TARGET_DEST_NOT_ALLOWED</p>	<p>Meaning: Program error. Certain commands are not allowed to be targeted to the same CPC and image on which the BCPii application is currently running. Such commands can cause the local system to be inoperable. Commands that cannot target the local CPC are:</p> <ul style="list-style-type: none"> • Hwi_Cmd_Activate • Hwi_Cmd_Activate_With_Actprof • Hwi_Cmd_Deactivate <p>Commands that cannot target the local image include:</p> <ul style="list-style-type: none"> • Hwi_Cmd_Activate_With_Actprof • Hwi_Cmd_Sysreset_IPLT <p>Commands that cannot target the local image (by itself or as a member of a user-defined image group) are:</p> <ul style="list-style-type: none"> • Hwi_Cmd_Activate • Hwi_Cmd_Deactivate • Hwi_Cmd_Load • Hwi_Cmd_PswRestart • Hwi_Cmd_Start • Hwi_Cmd_Stop • Hwi_Cmd_Sysreset • Hwi_Cmd_SCSI_Load • Hwi_Cmd_SCSI_Dump <p>Action: BCPii does not allow this command to be executed against the local CPC or local image. Validate the name of the target represented by the input connection token. If the target is correct, the command can only be issued from another CPC for a CPC-related command, or from another image for an image-related command.</p> <p>If the ConnectToken represents a user-defined image group, verify that the group does not contain the local image where this command is executing.</p>
<p>605 HWI_CMDPARAM_INACCESSIBLE</p>	<p>Meaning: Program error. The CmdParm data area cannot be accessed. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The CmdParm data area is either partially or completely not accessible by the application, or BCPii, or both. • The CmdParm data area can be too small. <p>Action: Check for probable coding error. Validate that the CmdParm_Ptr points to a data area where the CmdParm is and that the data area is accessible.</p>
<p>606 HWI_CMDTYPE_NOT_SUPPORTED</p>	<p>Meaning: The targeted hardware of the HWICMD request does not recognize the type of command being requested.</p> <p>Action: Verify that the targeted hardware is at a level that supports the type of command being issued.</p>

HWICMD

<p>607 HWI_CMD_NOT_SUPPORTED</p>	<p>Meaning: HWICMD is not supported with the current microcode level (MCL) installed on the target CPC, or the target CPC is at a lower hardware level than HWICMD supports (BCPii requires the target of an HWICMD to be at least at the z9 hardware level). The warning return code, HWI_CMD_NOT_SUPPORTED_WARNING, should have been returned on the previous HWICONN service call when the requested connect token was created to establish a connection to the CPC. See the return code section in “HWICONN — Establish a BCPii connection” on page 300 for more information.</p> <p>Action: Install the MCL that supports HWICMD on the target CPC or refrain from issuing HWICMD with a target older than the z9 hardware level. See the HWI_CMD_NOT_SUPPORTED_WARNING return code in the HWICONN section for the microcode level/engineering change (MCL/EC) that is required for HWICMD service call.</p>
<p>608 HWI_CMD_IMAGE_GROUP_IS_EMPTY</p>	<p>Meaning: Command did not execute because the connect token represents an image group that contains no images.</p> <p>Action: Ensure that the correct connect token was specified on the HWICMD request. If so, check with the SE/HMC engineer to determine the members that are in the group.</p>
<p>F00 HWI_NOT_AVAILABLE</p>	<p>Meaning: BCPii services are not available, and the system rejects the service request.</p> <p>Action: Notify the system programmer to start the BCPii address space and try the request again. See “Restarting the HWIBCPii address space” on page 269 about how to start the BCPii address space.</p> <p>Programs can also listen to ENF68 to determine when BCPii services are available. See <i>z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG</i> for how to listen for BCPii activation messages.</p>
<p>F01 HWI_AUTH_FAILURE</p>	<p>Meaning: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.</p> <p>Action: Check the calling program for a probable coding error.</p>

F02 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV. • Define control access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for a CPC or image group connection. • Define control access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for an image connection. • Define CONTROL access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for each image within the target image group for an image group connection. Note: It is possible that an application may have the proper authority to all images in a user-defined image group returned on a prior HWILIST invocation, yet still receive this error return code. This could be because HWILIST will only return image names that the user has the proper authority to view. In this case, it will be necessary to contact the HMC/SE administrator to find out if there are other image names contained in the user-defined image group that were not returned on the HWILIST invocation. Once these names have been acquired, the security administrator may be contacted to give CONTROL or higher access to these additional image names. • Ensure that the referenced Facility Class Profile is RACLIST-specified.
F03 HWI_INTERRUPT_STATUS_INV	<p>Meaning: The calling program is disabled. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWI_MODE_INV	<p>Meaning: The calling program is not in task mode. The system rejects this service request.</p> <p>Action: Check the calling program for a probable error.</p>
F05 HWI_LOCKS_HELD	<p>Meaning: The calling program is holding one or more locks. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F06 HWI_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects this service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again.</p>

HWICMD

F07 HWI_UNSUPPORTED_ENVIRONMENT	<p>Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine).</p> <p>Action: Issue the BCPii service from a different execution environment.</p>
FFF HWI_UNEXPECTED_ERROR	<p>Meaning: System error. The service that was called encountered an unexpected error. The system rejects the service call.</p> <p>Action: In many cases, BCPii has taken an abend to gather further diagnostic information. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Example

In the pseudocode example, the caller issues a call to activate an activation profile.

```
.  
.
CmdType = HWI_CMD_ACTIVATE;
HWI_CmdTypeParm.ForceType = HWI_CMD_Force;
CmdParm_Ptr = addr(HWI_CmdTypeParm);
CALL HWICMD (ReturnCode, ConnectToken, CmdType,
             CmdParm_Ptr, DiagArea)
.  
.
```

A REXX programming example for the HWICMD service:

Note: The command parm field names must exactly match the field names in the command parm structure declarations.

```
myCmdType = HWI_CMD_OSCMD                               /* oscmd */
myCmdParm.PriorityType = Hwi_Cmd_Priority
myCmdParm.OSCMDString = 'd a,1'
address bcpai
      "hwicmd RetCode myImgConnectToken myCmdType myCmdParm. myDiag."

If (RC <> 0) | (Retcode <> 0) Then
  Do
    Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
    If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
      Do
        Say ' Diag_index=' myDiag.DIAG_INDEX
        Say ' Diag_key=' myDiag.DIAG_KEY
        Say ' Diag_actual=' myDiag.DIAG_ACTUAL
        Say ' Diag_expected=' myDiag.DIAG_EXPECTED
        Say ' Diag_commerr=' myDiag.DIAG_COMMERR
        Say ' Diag_text=' myDiag.DIAG_TEXT
      End
    End
```

HWICONN — Establish a BCPii connection

Call the HWICONN service to establish a logical connection between the application and a central processor complex (CPC), a CPC image (LPAR), a capacity record, different types of activation profiles, or a user-defined image group. This facilitates subsequent services to perform operations that are related to that CPC, image, capacity record, activation profile, or a user-defined image group.

BCPii limits the total number of system-wide connections from all BCPii users to be no more than 4096 simultaneous connections.

Note: A connection remains active until one of the following occurs:

- A Disconnect service call (HWIDISC) has been invoked.
- A parent connection has been disconnected.
- A loss of connectivity to the associated CPC has been detected by BCPii.
- The address space of the caller has terminated.
- The current task of the caller has terminated if the connection has task affinity (TSO/E REXX or ISV-provided REXX execution environments).
- The BCPii address space has terminated.

Under normal circumstances, a connection remains active indefinitely. Since there are a finite number of total BCPii connections available in the entire system, a BCPii application should disconnect any BCPii connection it no longer needs.

Note: BCPii requires the FACILITY class to be RACLIST-specified. BCPii also automatically transforms the following to all uppercase characters when building the profile names passed to the security product: CPC, image, and caprec values pointed to by the ConnectTypeValue_Ptr.

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
Dispatchable unit mode:	Task
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:	Control parameters must be in the primary address space and addressable by the caller
Linkage:	Standard MVS linkage conventions are used

Programming requirements

See "Syntax, linkage and programming considerations" on page 271 for details about how to call BCPii services in the various programming languages.

REXX programming considerations for the HWICONN service

All information for the HWICONN service applies for REXX requests except:

- ConnectTypeValue replaces ConnectTypeValue_Ptr.

Restrictions

BCPii does not allow HWICONN to be issued from within a BCPii ENF exit routine.

Authorization

The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

The client application must also have at least one of the following access:

- Read access to the SAF-protected FACILITY class resource HWI.TARGET.netid.nau for HWI_CPC, HWI_RESET_ACTPROF, HWI_IMAGE_ACTPROF, HWI_LOAD_ACTPROF, or HWI_IMAGE_GROUP connections.
- Read access to the SAF-protected FACILITY class resource HWI.TARGET.netid.nau.imagename for HWI_IMAGE connections.
- Read access to the SAF-protected FACILITY class resource HWI.CAPREC.netid.nau.caprecid for HWI_CAPREC connections.

Note: BCPii requires the FACILITY class to be RACLIST-specified.

Syntax

Write the call as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Non-REXX parameters	REXX parameters
CALL HWICONN(ReturnCode, InConnectToken, OutConnectToken, ConnectType, ConnectTypeValue_Ptr, DiagArea);	address bcpii "hwiconn ReturnCode InConnectToken OutConnectToken ConnectType ConnectTypeValue DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

ReturnCode contains the return code from the service.

InConnectToken

Supplied parameter

- Type: Character string
- Length: 16 bytes

InConnectToken represents a connect token that was returned by a previous HWICONN HWI_CPC invocation. For image, capacity record, activation profile, and user-defined image group connections, the input connection token must represent an active CPC connection.

In most cases, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call. For BCPii REXX execs running under TSO/E or ISV-provided REXX environments, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same task as this service call.

InConnectToken is not relevant to a connect type of HWI_CPC, and it is ignored.

OutConnectToken

Returned parameter

- Type: Character string
- Length: 16 bytes

OutConnectToken returns a connect token that uniquely represents a connection to BCPii. This parameter can be used as input on subsequent BCPii invocations to identify which connection the service wants to communicate.

A connect token returned for an HWI_CPC connection can be specified on subsequent services to perform operations against this particular CPC, or on a subsequent HWICONN as the InConnectToken parameter when attempting a connection to a particular image (LPAR), capacity record (CAPREC), or activation profile.

Likewise, a connect token returned for an HWI_IMAGE or HWI_CAPREC connection can be specified on subsequent services to perform operations against this particular image (LPAR) or capacity record (CAPREC) respectively.

A connect token returned for an HWI_RESET_ACTPROF, HWI_IMAGE_ACTPROF, or HWI_LOAD_ACTPROF connection can be specified on subsequent HWIQUERY or HWISET service calls to query or set specific values associated with the specified Reset, image, or Load activation profile respectively.

A connection token returned for an HWI_IMAGE_GROUP can be specified on a subsequent HWIQUERY service call to query values associated with the group profile, on a subsequent HWICMD service call to issue commands to all members in the image group, or on a subsequent HWILIST service call to list the images in the image group.

ConnectType

Supplied parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

ConnectType specifies the type of connection to be established.

Constant in Hexadecimal (Decimal) Equate Symbol	Description
1 (1) HWI_CPC	Requests to establish a connection to a target CPC that the application is to communicate with.
2 (2) HWI_IMAGE	Requests to establish a connection to an image of a CPC that the application is to communicate with. The input connection token must represent an active CPC connection.
3 (3) HWI_CAPREC	Requests to establish a connection to a capacity record of a CPC that the application is to communicate with. The input connection token must represent an active CPC connection.

HWICONN

Constant in Hexadecimal (Decimal) Equate Symbol	Description
4 (4) HWI_RESET_ACTPROF	Requests to establish a connection to a reset activation profile associated with a particular CPC. The input connection token must represent an active CPC connection.
5 (5) HWI_IMAGE_ACTPROF	Requests to establish a connection to an image activation profile associated with a particular CPC. The input connection token must represent an active CPC connection.
6 (6) HWI_LOAD_ACTPROF	Requests to establish a connection to a load activation profile associated with a particular CPC. The input connection token must represent an active CPC connection.
7 (7) HWI_IMAGE_GROUP	Requests to establish a connection to a user-defined image group on a particular CPC. The input connection token must represent an active CPC connection. Note: This ConnectType is only available when targeting a z10 or higher CPC.

ConnectTypeValue_Ptr (non-REXX)

ConnectTypeValue (REXX)

Supplied parameter

- Type: Pointer (non-REXX), character (REXX)
- Length: 4 bytes (non-REXX)

Non-REXX:

ConnectTypeValue_Ptr specifies the address of the name of the requested target to be connected to. The type of connection determines the value required.

REXX:

ConnectTypeValue is the name of the requested target to be connected to. The type of connection determines the value required.

Connect Types	Values to be specified
HWI_CPC	<ul style="list-style-type: none"> • A 17-character network address (sometimes referred to as the SNA address) that uniquely represents a CPC in the attached process control network. The network address should be in the form of a 1- through 8-character network identifier (<i>netid</i>), followed by a period, and then followed by a 1- through 8-character network addressable unit (NAU) name. The network address should be padded with trailing blanks if the total string length of the network address is less than 17 characters. Note: Both the <i>netid</i> and NAU name must consist of alphanumeric characters (0-9, A-Z). Example: <code>net1.cpc01</code> • An <code>*</code> is a special value that can also be specified with this ConnectType. If specified, this allows the application to connect to the local host CPC without having to know the network address of the local host CPC (<i>netid.nau</i>). <p>Note: An HWILIST HWI_LIST_CPCS operation returns a list of CPCs available to be connected to in the form of <i>netid.nau</i>.</p>
HWI_IMAGE	<p>An 8-character image name padded with trailing blanks. Note: The LPAR name is a 1- through 8-alphanumeric (0-9, A-Z) character name that must have an alphabetic first character. Special characters (\$, #, @), although currently allowed, are being reserved for future use. See <i>PR/SM Planning Guide</i> for details.</p>
HWI_CAPREC	<p>An 8-character capacity record (CAPREC) name padded with trailing blanks. Note: The CAPREC name is a 1- to 8-alphanumeric (0-9, A-Z, a-z) character name.</p>

Connect Types	Values to be specified
HWI_RESET_ACTPROF	A 16-character alphanumeric (0-9, A-Z) reset activation profile name padded with trailing blanks.
HWI_IMAGE_ACTPROF	A 16-character alphanumeric (0-9, A-Z) image activation profile name padded with trailing blanks.
HWI_LOAD_ACTPROF	A 16-character alphanumeric (0-9, A-Z) load activation profile name padded with trailing blanks.
HWI_IMAGE_GROUP	A 30 character null-terminated image group name.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the DiagArea might not be filled in, and the data returned in the area should be ignored.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value that is specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code that is returned from the console application API or the BCPii transport layer.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, “BCPii communication error reason codes,” on page 583 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0002yyyy' because of one of the following reasons:

Table 62. Reasons for abend X'042', RC X'0002yyyy'

yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a

different reason code may result. See *z/OS MVS System Codes* for additional information.

Return codes

Return Code in Hexadecimal Equate Symbol	Meaning and Action
0 HWI_OK	<p>Meaning: Successful completion.</p> <p>Action: None.</p>
4 HWI_CMD_NOT_SUPPORTED_WARNING	<p>Meaning: Successful completion. This warning return code is informational.</p> <p>The target CPC being connected to has a microcode level (MCL) that does not support HWICMD, or the target CPC is at a lower hardware level than HWICMD supports (BCPii requires the target of an HWICMD to be at least at the z9 hardware level). If a subsequent HWICMD is issued with this returned connect token, the call will be rejected with a return code of HWI_CMD_NOT_SUPPORTED.</p> <p>Action: Install the MCL/EC that supports HWICMD for the target CPC. The required MCL/EC are G40965.133 for a z9 CPC, and F85906.116 for a z10 CPC.</p>
100 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified input connection token is not valid. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The input connection token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken. • The input connection token does not represent an active connection. The connection specified might have already been disconnected by the HWIDISC service call, or have been implicitly disconnected by BCPii because of loss of connectivity with the target CPC. • The input connection token is not associated with the address space of the caller. The InConnectToken specified is associated with a different address space than the caller of this service call. <p>Action: Check for probable coding error.</p>

Return Code in Hexadecimal Equate Symbol	Meaning and Action
101 HWI_COMMUNICATION_ERROR	<p>Meaning: A communication error is detected. The hardware management console application API (HWMCA) or the BCPii transport layer has returned with a failing return code.</p> <p>Action: See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPii transport layer.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, "BCPii communication error reason codes," on page 583.</p>
102 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field.</p>
103 HWI_CONNECT_TOKEN_INACTIVE	<p>Meaning: The specified connect token is no longer valid. The connection has been disconnected or it is in the progress of being disconnected.</p> <p>Action: Check for probable coding error. Verify that the specified connect token is still active. If connectivity to the targeted CPC connection no longer exists, all connections associated with that CPC will no longer have a connect token that can be used.</p>
201 HWI_CONNTYPE_INV	<p>Meaning: Program error. The connection type specified in the call is not valid. The system rejects the service call.</p> <p>Action: Check for probable coding error. Validate that the conntype value passed to the service is one of the accepted values.</p>
202 HWI_CONNTYPE_VALUE_INV	<p>Meaning: Program error. The connection name specified in the call is not valid. The specified connection name is not syntactically valid, it does not exist, or it is currently not available. The system rejects the service call.</p> <p>Action: Check for probable coding error. Verify that the connection name is syntactically correct, valid in the current HMC configuration, and currently available.</p>

HWICONN

Return Code in Hexadecimal Equate Symbol	Meaning and Action
203 HWI_CONNTYPE_VALUE_INACCESSIBLE	<p>Meaning: Program error. The connection type value data area is either partially or completely inaccessible by the application, or the Base Control Program internal interface (BCPii) address space, or both.</p> <p>Action: Check for probable coding error. Verify that the ConnectTypeValue_Ptr points to a data area where the connect type value is, and make sure that the data area is accessible.</p>
204 HWI_MAX_CONNECTIONS_REACHED	<p>Meaning: The number of connections has reached the maximum number of system-wide connections (4096) that BCPii permits, or BCPii has run out of system resources to satisfy the HWICONN request, or both.</p> <p>Action: Disconnect connections that are no longer needed, and try the request again.</p>
205 HWI_CONNTYPE_NOT_SUPPORTED	<p>Meaning: The targeted hardware of the HWICONN request does not support the connect type specified.</p> <p>Action: Verify that the targeted hardware supports the type of request being made.</p>
F00 HWI_NOT_AVAILABLE	<p>Meaning: BCPii services are not available, and the system rejects the service request.</p> <p>Action: Notify the system programmer to start the BCPii address space and try the request again. See “Restarting the HWIBCPii address space” on page 269 about how to start the BCPii address space.</p> <p>Programs can also listen to ENF68 to determine when BCPii services are available. See <i>z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG</i> for how to listen for BCPii activation messages.</p>
F01 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.</p> <p>Action: Check the calling program for a probable coding error.</p>

Return Code in Hexadecimal Equate Symbol	Meaning and Action
F02 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV. • Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for a CPC, activation profile, or image group connection. • Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for an image connection. • Define read access authorization to the FACILITY class resource profile HWI.CAPREC.netid.nau.caprecid for a capacity record connection. • Ensure that the referenced Facility Class Profiles are RACLIST-specified. • For CPC connections only: The SNMP community name specified in the security product (SAF) for a particular target CPC does not match the SNMP community name defined in the support element of the target CPC. See “Community name defined in the security product for each CPC” on page 263 for further information regarding community name setup.
F03 HWI_INTERRUPT_STATUS_INV	<p>Meaning: The calling program is disabled. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWI_MODE_INV	<p>Meaning: The calling program is not in task mode. The system rejects this service request.</p> <p>Action: Check the calling program for a probable error.</p>
F05 HWI_LOCKS_HELD	<p>Meaning: The calling program is holding one or more locks. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Return Code in Hexadecimal Equate Symbol	Meaning and Action
F06 HWI_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects this service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again.</p>
F07 HWI_UNSUPPORTED_ENVIRONMENT	<p>Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine).</p> <p>Action: Issue the BCPii service from a different execution environment.</p>
FFF HWI_UNEXPECTED_ERROR	<p>Meaning: System error. The service that was called encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Example

In the pseudocode example, the application attempts to establish a connection between the application and the target CPC.

```
.
.
InConnectToken = 16blanks;
ConnectType = HWI_CPC;
ConnectTypeValue_Ptr = Addr(ConnectTypeValue);
ConnectTypeValue = 'CPCPLEX1.CPC01';
CALL HWICONN (ReturnCode, InConnectToken, OutConnectToken,
              ConnectType, ConnectTypeValue_Ptr, DiagArea)
(After the call, OutConnectToken contains a token that can be used on all
subsequent calls to perform CPC functions against the 'CPCPLEX1.CPC01' CPC
including connecting to images, capacity records, and activation profiles
residing on the CPC.)
.
.
```

A REXX programming example for the HWICONN service:

```
myConnectType      = HWI_CPC           /* CPC connect type */
myConnectTypeValue = 'IBM390xx.H123   ' /* 17-char CPC name */

address bcpii
      "hwiconn Retcode myInConnectToken myOutConnectToken myConnectType
      myConnectTypeValue myDiag."

If (RC <> 0) | (Retcode <> 0) Then
  Do
    Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
    If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
      Do
        Say ' Diag_index=' myDiag.DIAG_INDEX
        Say ' Diag_key=' myDiag.DIAG_KEY
        Say ' Diag_actual=' myDiag.DIAG_ACTUAL
```

```

        Say ' Diag_expected=' myDiag.DIAG_EXPECTED
        Say ' Diag_commerr=' myDiag.DIAG_COMMERR
        Say ' Diag_text=' myDiag.DIAG_TEXT
    End
End

```

HWIDISC — Release a BCPii connection

Call the HWIDISC service to release the logical connection between the application and the identified CPC, image, capacity record, different types of activation profiles, or user-defined image groups. If the connect token represents a CPC, any subordinate image, capacity record, activation profile, or user-defined image group connection associated with the same CPC connection is also released.

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
Dispatchable unit mode:	Task
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:	Control parameters must be in the primary address space and addressable by the caller
Linkage:	Standard MVS linkage conventions are used

Programming requirements

See “Syntax, linkage and programming considerations” on page 271 for details about how to call BCPii services in the various programming languages.

REXX programming considerations for the HWIDISC service

All information for the HWIDISC service applies for REXX requests except:

- In the System REXX environment, BCPii connections are associated with the address space that issued the AXREXX macro service call. When this address space terminates, BCPii will implicitly disconnect the connection.
- In the TSO/E and ISV-provided REXX environments, BCPii connections are associated with the current running task. When this task terminates, BCPii will implicitly disconnect the connection.

Restrictions

BCPii does not allow HWIDISC to be issued from within a BCPii ENF exit routine.

Authorization

The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

The client application must also have at least read access to the following class resources:

HWIDISC

- The SAF-protected FACILITY class resource HWI.TARGET.*netid.nau* for HWI_CPC, HWI_RESET_ACTPROF, HWI_IMAGE_ACTPROF, HWI_LOAD_ACTPROF, or HWI_IMAGE_GROUP connections.
- The SAF-protected FACILITY class resource HWI.TARGET.*netid.nau.imagename* for HWI_IMAGE connections.
- The SAF-protected FACILITY class resource HWI.CAPREC.*netid.nau.caprecid* for HWI_CAPREC connections.

Note: BCPii requires the FACILITY class to be RACLIST-specified.

Syntax

Write the call as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Non-REXX parameters	REXX parameters
CALL HWIDISC(ReturnCode, ConnectToken, DiagArea);	address bcpii "hwidisc ReturnCode ConnectToken DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

ReturnCode contains the return code from the service.

ConnectToken

Supplied parameter

- Type: Character string
- Length: 16 bytes

ConnectToken specifies the logical connection to be released. A ConnectToken represents a logical connection between the application and a CPC, image, capacity record, activation profile, or user-defined image group and is returned as an output parameter on the HWICONN service call.

In most cases, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call. For BCPii REXX execs running under the TSO/E or ISV-provided REXX environments, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same task.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the DiagArea might not be filled in, and the data returned in the area should be ignored.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code that is returned from the console application API or the BCPii transport layer.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, “BCPii communication error reason codes,” on page 583 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0003yyyy' because of one of the following reasons:

Table 63. Reasons for abend X'042', RC X'0003yyyy'

yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See *z/OS MVS System Codes* for additional information.

Return codes

When the service returns control to the caller, GPR 15 and ReturnCode contain a hexadecimal return code.

Return Code in Hexadecimal Equate Symbol	Meaning and Action
0 HWI_OK	Meaning: Successful completion. Action: None.

Return Code in Hexadecimal Equate Symbol	Meaning and Action
100 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified connect token is not valid. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The input connection token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken. • The connect token does not represent an active connection. The connection specified might have already been disconnected using the HWIDISC service call. • The connect token is not associated with the address space of the caller. The ConnectToken specified is associated with a different address space than the caller of this service call. <p>Action: Check for probable coding error.</p>
101 HWI_COMMUNICATION_ERROR	<p>Meaning: A communication error is detected. The hardware management console application API (HWMCA) or the BCPii transport layer has returned with a failing return code.</p> <p>Action: See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPii transport layer.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, "BCPii communication error reason codes," on page 583.</p>
102 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field.</p>
901 HWI_DISC_INPROGRESS	<p>Meaning: Another Disconnect request is already in progress. This request is redundant.</p> <p>Action: None.</p>

Return Code in Hexadecimal Equate Symbol	Meaning and Action
F00 HWI_NOT_AVAILABLE	<p>Meaning: BCPii services are not available, and the system rejects the service request.</p> <p>Action: Notify the system programmer to start the BCPii address space and try the request again. See “Restarting the HWIBCPii address space” on page 269 about how to start the BCPii address space.</p> <p>Programs can also listen to ENF68 to determine when BCPii services are available. See <i>z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG</i> for how to listen for BCPii activation messages.</p>
F01 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV. • Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for a CPC, activation profile, or image group connection. • Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for an image connection. • Define read access authorization to the FACILITY class resource profile HWI.CAPREC.netid.nau.caprecid for a capacity record connection. • Ensure that the referenced Facility Class Profiles are RACLIST-specified.
F03 HWI_INTERRUPT_STATUS_INV	<p>Meaning: The calling program is disabled. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWI_MODE_INV	<p>Meaning: The calling program is not in task mode. The system rejects this service request.</p> <p>Action: Check the calling program for a probable error.</p>

Return Code in Hexadecimal Equate Symbol	Meaning and Action
F05 HWI_LOCKS_HELD	<p>Meaning: The calling program is holding one or more locks. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F06 HWI_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects this service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again.</p>
F07 HWI_UNSUPPORTED_ENVIRONMENT	<p>Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine).</p> <p>Action: Issue the BCPii service from a different execution environment.</p>
FFF HWI_UNEXPECTED_ERROR	<p>Meaning: System error. The service that was called encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Example

In the pseudocode example, the caller issues a call to release a connection between the application and a CPC.

```

:
:
CALL HWIDISC (ReturnCode, ConnectToken, DiagArea)
:
:

```

A REXX programming example for the HWIDISC service:

```

address bcpii
    "hwidisc Retcode myConnectToken myDiag."

If (RC <> 0) | (Retcode <> 0) Then
    Do
        Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
        If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
            Do
                Say ' Diag_index=' myDiag.DIAG_INDEX
                Say ' Diag_key=' myDiag.DIAG_KEY
                Say ' Diag_actual=' myDiag.DIAG_ACTUAL
                Say ' Diag_expected=' myDiag.DIAG_EXPECTED
                Say ' Diag_commerr=' myDiag.DIAG_COMMERR
                Say ' Diag_text=' myDiag.DIAG_TEXT
            End
        End
    End

```

HWIEVENT — Register or unregister for BCPii events

Call the HWIEVENT service for the following purposes:

1. Register an application and its connection to receive notification of:
 - One or more hardware or software events occurring on the connected CPC or image.
 - Communication errors between BCPii and the connected CPC or image.
2. Delete the registration for one or more previously registered events.

Monitoring events occurring on a particular CPC or image

For hardware and software events, an application can register with BCPii to be notified when an event occurs for the targeted CPC or image. Under the covers, BCPii communicates the registration request with the support element (SE) of the targeted CPC or image if necessary and also registers the user-provided exit with the Event Notification Facility (ENF). When the event occurs on the targeted CPC or image, BCPii receives notification and signals the appropriate ENF68. The user's exit receives control with data unique for the event that just occurred. The data mapping for these different events can be found in the public interface files shipped with BCPii (HWICIC for the C programming language, HWICIREX for the REXX programming language, and HWICIASM for the assembler programming language). BCPii also provides a sample of an ENF event exit in SYS1.SAMPLIB (HWIXMCX1) that can be a good starting point for coding a BCPii ENF exit.

Note: BCPii user-defined image groups are a powerful way to issue commands to all members of a group simultaneously. Commands targeted to a user-defined image group will result in one image command response event being generated for each image in the image group. If event notification is desired for an image in an image group, register the image for the command response event to enable delivery of the event to the BCPii ENF exit.

Monitoring operating system message events (Hwi_Event_OpSysMsg)

Your application can monitor all operating system messages appearing on a z/OS console by using the HWIEVENT service to register for the EventIDs parameter value Hwi_Event_OpSysMsg.

For the majority of messages issued on the image being monitored, a single BCPii operating system message event will contain the entire message in the returned event data (HWIENF68 data mapping).

For messages that are larger than approximately 3000 bytes, it is possible that the operating system message is longer than the architected maximum buffer size allowed by the communications protocol used by both the z/OS consoles component and BCPii to communicate with the support element. As a result, BCPii delivers these single large messages in multiple operating system message events. Each of these operating system message events representing a single large message will have the same values in the HWIENF68 data mapping for the msgId, msgDate, and msgTime fields. An application can determine that all of the operating system message events have been delivered for the single large message by consulting the msgId of a subsequent message event. If it has changed from the previous msgId, the operating system message event represents a new operating system message.

Monitoring communication availability between BCPii and the CPC

While not common, BCPii may occasionally experience communication delays or interruptions of service between itself and the targeted CPC and its associated support element. BCPii provides a mechanism through its BCPii communication error class of events to detect these interruptions and to allow an application to know when these interruptions of service have been resolved.

BCPii keeps a heartbeat between itself and each CPC where its applications desire connectivity. If BCPii fails to receive its regular heartbeat from an SE associated with a CPC, BCPii attempts a communication flow to this SE. If the SE responds successfully to this communication attempt by BCPii, BCPii signals a *temporary communication error*, (ENF QUAL value 02010001), meaning that the reason for the heartbeat not being received is not known, but the communication path between BCPii and the SE seems to be operational at this time. During the past few minutes, one or more events may have been lost.

If the SE does not respond to the BCPii communication attempt, BCPii assumes that there is a serious communication problem and signals a *permanent communication error*, (ENF QUAL value 02010002). At this point, no HWIEVENT or HWICMD API requests to this CPC are processed by BCPii and no event delivery take place for events registered on this CPC and its images. BCPii closes its internal connections with the CPC and cleans up resources associated with command processing and event delivery to and from this CPC.

BCPii then regularly attempts to restart its command processing and event delivery connections to this CPC. When this connection to the CPC has been re-established, BCPii signals a *communication available event*, (ENF QUAL value 02010003). At this point, applications currently having valid connections to this CPC and its images are allowed to use the HWIEVENT and HWICMD APIs to the CPC and its images. Receipt of events originating from the CPC and its images commence once again.

An application may choose to register for these communication availability events via the HWIEVENT ADD service (EventIDs parameter value Hwi_Event_HwCommError), or it may choose to use the ENFREQ LISTEN macro to listen for these events apart from any specific BCPii connection.

Monitoring the status of the BCPii address space

An application can monitor the status of the BCPii address space itself by using the ENFREQ LISTEN service and specifying the appropriate QUAL values to monitor when the BCPii address space becomes active and when it terminates:

- BCPii signals an ENF68 with a QUAL value of 01000002 when the BCPii address space becomes active.
- BCPii signals an ENF68 with a QUAL value of 01000001 when the BCPii address space becomes unavailable.

While it is possible to use the HWIEVENT service to allow an application to register for the Hwi_Event_BCPiiStatus event, this is not a recommended way to monitor initialization or termination of the BCPii address space. When the BCPii address space terminates, BCPii asynchronously asks the system to delete all ENF registrations made on behalf of applications that have issued HWIEVENT Add requests. If the deletion of the ENF registration occurs prior to the BCPii address space termination, the ENF exit will no longer receive control when BCPii signals that it is down.

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
Dispatchable unit mode:	Task
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:	Control parameters must be in the primary address space and addressable by the caller
Linkage:	Standard MVS linkage conventions are used

Programming requirements

See “Syntax, linkage and programming considerations” on page 271 for details about how to call BCPii services in the various programming languages. For programming language C, see “Restrictions.”

See “HWIEVENT” on page 598 for the summary table of the BCPii HWIEVENT types and the objects that can be registered or unregistered for each event.

REXX programming considerations for the HWIEVENT service

All information for the HWIEVENT service applies for REXX requests except:

- EventIDs is a 32-element stem-variable representing all of the event bits as defined in the HWICIREX include file.
- Because the Event Notification Facility (ENF) does not support REXX exits, the caller must provide the address of a non-REXX ENF exit routine.
- The EventExitAddr must be specified as the 8-character representation of a 4-byte hexadecimal value.

Restrictions

- This service is not used by C language callers running in a z/OS UNIX System Services environment. See “HWIManageEvents — Manage the list of BCPii events” on page 406.
- BCPii does not allow HWIEVENT to be issued from within a BCPii ENF exit routine.
- BCPii does not allow HWIEVENT to be issued from a REXX exec running in the TSO/E or ISV-provided REXX environments.

Authorization

The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

The client application must have at least read access to the SAF-protected FACILITY class resource HWI.TARGET.*netid.nau* for a ConnectToken representing a CPC connection, or HWI.TARGET.*netid.nau.imagename* for ConnectToken representing an image connection.

HWIEVENT

Note: BCPii requires the FACILITY class to be RACLIST-specified.

Syntax

Write the call as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Non-REXX parameters	REXX parameters
CALL HWIEVENT(ReturnCode, ConnectToken, EventAction, EventIDs, EventExitMode, EventExitAddr, EventExitParm, DiagArea);	address bcpii "hwievent ReturnCode ConnectToken EventAction EventIDs. EventExitMode EventExitAddr EventExitParm DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

ReturnCode contains the return code from the service.

ConnectToken

Supplied parameter

- Type: Character string
- Length: 16 bytes

ConnectToken represents a logical connection between the application and a CPC or image. The ConnectToken is an output parameter on the HWICONN service call.

The ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call.

EventAction

Supplied parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

EventAction specifies the type of action for the service.

Constant in Hexadecimal (Decimal) Equate Symbol	Description
1 (1) HWI_EVENT_ADD	Registers to be notified when the requested events occur.
2 (2) HWI_EVENT_DELETE	Deletes the registration for notification.

EventIDs (non-REXX)**EventIDs. (REXX)**

Supplied parameter

- Type: Integer (non-REXX), stem variable (REXX)
- Length: 128 bits (16 bytes) (non-REXX)

EventIDs specifies the events to be added or deleted.

Non-REXX:

Each event is a 1-bit field from bit position 97 to 128 in this data area. If the bit is on, the service performs the EventAction operation for the event on the requested connection.

REXX:

Each event is represented by an IBM-supplied EventIDs tail label or tail value constant. If the value is on, the service performs the EventAction operation for the event on the requested connection.

It is recommended to use the IBM-supplied EventIDs tail labels defined in HWICIREX.

Note: A single connection may not register for a particular event more than once.

The following event IDs or tail labels can be specified:

EventIDs (non-REXX) / tail label for EventIDs stem (REXX)	Bit position in structure specified on EventIDs (non-REXX)	Tail value constant of the user-defined EventIDs stem (REXX)	Description
Hwi_EventID_EyeCatcher	1-96	N/A	Control block identifier. Note: HWI_EVENTID_TEXT can be used to initialize this field.
Hwi_Event_CmdResp	97	1	Requests to add or delete the registration for notification of the command response events. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
Hwi_Event_StatusChg	98	2	Requests to add or delete the registration for notification of the status change events. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
Hwi_Event_NameChg	99	3	Requests to add or delete the registration for notification of the object name change events. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
Hwi_Event_ActProfChg	100	4	Requests to add or delete the registration for notification of the change events for the activation profile name. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
Hwi_Event_ObjCreate	101	5	Requests to add or delete the registration for notification of the object created events. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
Hwi_Event_ObjDestroy	102	6	Requests to add or delete the registration for notification of the object destroyed (deleted) events. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .

HWIEVENT

EventIDs (non-REXX) / tail label for EventIDs stem (REXX)	Bit position in structure specified on EventIDs (non-REXX)	Tail value constant of the user-defined EventIDs stem (REXX)	Description
Hwi_Event_ObjException	103	7	Requests to add or delete the registration for notification of the exception state events. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
Hwi_Event_ApplStarted	104	8	Requests to add or delete the registration for notification of the console application started events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_ApplEnded	105	9	Requests to add or delete the registration for notification of the console application ended events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_HwMsg	106	10	Requests to add or delete the registration for notification of the hardware message events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_HwMsgDel	107	11	Requests to add or delete the registration for notification of the hardware message deletion events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_SecurityEvent	108	12	Requests to add or delete the registration for notification of the support element (SE) console security events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_CapacityChg	109	13	Requests to add or delete the registration for notification of the capacity change events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_CapacityRecord	110	14	Requests to add or delete the registration for notification of the capacity record change events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_OpSysMsg	111	15	Requests to add or delete the registration for notification of the operating system message events. Note: The input connection token must only represent an <i>image connection</i> .
Hwi_Event_HwCommError	112	16	Requests to add or delete the registration for notification of the hardware communication error events. Note: The input connection token must only represent a <i>CPC connection</i> .
Hwi_Event_BCPIIStatus	113	17	Requests to add or delete the registration for notification of BCPII status change events. Note: This method is not recommended for determining if the BCPII address space becomes available or unavailable. See the description of the HWIEVENT service for more information.
Hwi_Event_DisabledWait	114	18	Requests to add or delete the registration for notification of disabled wait events. Note: The input connection token must only represent an <i>image connection</i> .

EventIDs (non-REXX) / tail label for EventIDs stem (REXX)	Bit position in structure specified on EventIDs (non-REXX)	Tail value constant of the user-defined EventIDs stem (REXX)	Description
Hwi_Event_PowerChange	115	19	Requests to add or delete the registration for notification of any power characteristics change events. Note: The input connection token must represent a <i>CPC connection</i> .
Hwi_Event_Reserved	116-128	N/A	Reserved, must be initialized to binary zeros.

EventExitMode

Supplied parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

EventExitMode specifies the type of the exit mode for the service.

Constant in Hexadecimal (Decimal) Equate Symbol	Description
1 (1) HWI_EVENT_TASK	The base control program internal interface gives control in task mode to an ENF listen-exit routine as specified on the EventExitAddr parameter. Task mode ENF exits must reside in common storage.

At present, only one value is allowed for this parameter. In the future, IBM might choose to allow additional values to be specified.

EventExitAddr

Supplied parameter

- Type: Pointer (non-REXX), character representation of a pointer (REXX)
- Length: 4 bytes (non-REXX), 8 characters (REXX)

EventExitAddr specifies the address of an ENF listen-exit routine that receives control when the requested event occurs. The application is responsible for writing this ENF exit routine, as described in the ENFREQ documentation for ENF 68 found in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*. For further information regarding the coding of ENF exits, see the "Listening for System Events" chapter in the *z/OS MVS Programming: Authorized Assembler Services Guide*.

EventExitParm

Supplied parameter

- Type: Pointer or integer (non-REXX), character representation of a pointer or integer (REXX)
- Length: 4 bytes (non-REXX), up to 8 numeric characters (REXX)

EventExitParm specifies an optional value to be passed to the ENF listen-exit when invoked, as described in the ENFREQ documentation for ENF 68 found in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*.

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the DiagArea can contain further

HWIEVENT

information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the DiagArea might not be filled in, and the data returned in the area should be ignored.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The return code that is returned from the console application API or the BCPii transport layer.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, "BCPii communication error reason codes," on page 583 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0004yyyy' because of one of the following reasons:

Table 64. Reasons for abend X'042', RC X'0004yyyy'

yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See *z/OS MVS System Codes* for additional information.

Return codes

When the service returns control to the caller, GPR 15 and ReturnCode contain a hexadecimal return code.

Return Code in Hexadecimal Equate Symbol	Meaning and Action
0 HWI_OK	Meaning: Successful completion. Action: None.

Return Code in Hexadecimal Equate Symbol	Meaning and Action
100 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified connect token is not valid. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The connect token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken. • The connect token does not represent an active connection. The connection specified might have already been disconnected by the HWIDISC service call. • The connect token is not associated with the address space of the caller. The ConnectToken specified is associated with a different address space than the caller of this service call. <p>Action: Check for probable coding error.</p>
101 HWI_COMMUNICATION_ERROR	<p>Meaning: A communication error is detected. The hardware management console application API (HWMCA) or the BCPii transport layer has returned with a failing return code.</p> <p>Action: See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPii transport layer. BCPiis CTRACE might provide further diagnostic information if the problem can not easily be resolved. See <i>z/OS MVS System Commands</i> for further information about starting and stopping CTRACE.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, "BCPii communication error reason codes," on page 583.</p>
102 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field.</p>
103 HWI_CONNECT_TOKEN_INACTIVE	<p>Meaning: The specified connect token is no longer valid. The connection has been disconnected, or it is in the progress of being disconnected.</p> <p>Action: Check for probable coding error. Verify that the specified connect token is still active. If connectivity to the targeted CPC connection no longer exists, all connections associated with that CPC will no longer have a connect token that can be used.</p>

HWIEVENT

Return Code in Hexadecimal Equate Symbol	Meaning and Action
701 HWI_EVENT_EXITMODE_INV	<p>Meaning: Program error. The requested EventExitMode on the call is not valid. The system rejects the service call.</p> <p>Action: Check for probable coding error.</p>
702 HWI_EVENT_EXITADDR_INV	<p>Meaning: Program error. The requested EventExitAddr on the call is not valid. The system rejects the service call.</p> <p>Action: Check for probable coding error.</p>
703 HWI_EVENT_ACTION_INV	<p>Meaning: Program error. The requested EventAction on the call is not valid. The system rejects the service call.</p> <p>Action: Check for probable coding error.</p>
704 HWI_EVENT_IDS_INV	<p>Meaning: Program error. The requested EventIDs on the call is not valid. The system rejects the service call. This return code indicates one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The first 12 bytes of the EventIDs parameter is not equal to the expected Eyecatcher of HWIEVENTBLCK (non-REXX only). • The reserved area of the EventIDs parameter contains a non-zero value. • The EventIDs specified applies only to a CPC connection, but the ConnectToken specified represents an image or capacity record connection. • The EventIDs specified applies only to image connections, but the ConnectToken specified represents a CPC or capacity record connection. • A request which specified an EventAction of HWI_EVENT_DELETE also specified EventIDs of one or more events that were not registered on a previous HWIEVENT EventAction = HWI_EVENT_ADD request for the connection. <p>Action: Check for probable coding error.</p>
F00 HWI_NOT_AVAILABLE	<p>Meaning: BCPii is not available, and the system rejects the service request.</p> <p>Action: Notify the system programmer to start the BCPii address space and try the request again. See “Restarting the HWIBCPii address space” on page 269 about how to start the BCPii address space.</p> <p>Programs can also listen to ENF68 to determine when BCPii services are available. See <i>z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG</i> for how to listen for BCPii activation messages.</p>

Return Code in Hexadecimal Equate Symbol	Meaning and Action
F01 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV. • Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for CPC connection. • Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for an image connection. • Ensure that the referenced FACILITY class profiles are RACLIST-specified.
F03 HWI_INTERRUPT_STATUS_INV	<p>Meaning: The calling program is disabled. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWI_MODE_INV	<p>Meaning: The calling program is not in task mode. The system rejects this service request.</p> <p>Action: Check the calling program for a probable error.</p>
F05 HWI_LOCKS_HELD	<p>Meaning: The calling program is holding one or more locks. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F06 HWI_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects this service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again.</p>

HWIEVENT

Return Code in Hexadecimal Equate Symbol	Meaning and Action
F07 HWI_UNSUPPORTED_ENVIRONMENT	<p>Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine).</p> <p>Action: Issue the BCPii service from a different execution environment.</p>
FFF HWI_UNEXPECTED_ERROR	<p>Meaning: System error. The service that was called encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Example

In the pseudocode example, the caller issues a call to register to be notified when the command response events and status change events occur.

```

Declare (ReturnCode, EventAction, EventExitMode) Fixed(31);
Declare ConnectToken Isa(HWI_CONNTOKEN_TYPE);
Declare EventIDs Isa(HWI_EVENTIDS_TYPE);
Declare (EventExitAddr, EventExitParm) Ptr(31);
Declare DiagArea Isa(HWI_DIAGAREA_TYPE);
Declare EventExit Entry External;

EventAction = HWI_EVENT_ADD;
Hwi_EventID_EyeCatcher = HWI_EVENTID_TEXT;
Hwi_Event_CmdResp = on;
Hwi_Event_StatusChg = on;
Hwi_Event_Reserved = 0;
EventExitMode = HWI_EVENT_TASK;
EventExitAddr = ADDR(EventExit);
EventExitParm = 0;

CALL HWIEVENT (ReturnCode, ConnectToken, EventAction, EventIDs,
              EventExitMode, EventExitAddr, EventExitParm, DiagArea);

```

A REXX programming example for the HWIEVENT service:

```

myAction = HWI_EVENT_ADD
myEventIDs. = 0 /*Initialize all EventIDs to 0 */
myEventIDs.Hwi_Event_CmdResp = 1
myEventIDs.Hwi_Event_StatusChg = 1
myEventIDs.Hwi_Event_ActProfChg = 1

myMode = HWI_EVENT_TASK
myEventExitAddr = 0F123456 /* char rep of 4 byte hex address */
myEventExitParm = 0

address bcp ii
"hwievent RetCode myConnectToken myEventAction myEventIDs. myEventExitMode
 myEventExitAddr myEventExitParm myDiag."

If (RC <> 0) | (Retcode <> 0) Then
  Do
    Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
    If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
      Do
        Say ' Diag_index=' myDiag.DIAG_INDEX
        Say ' Diag_key=' myDiag.DIAG_KEY
      End
    End

```

```

Say ' Diag_actual=' myDiag.DIAG_ACTUAL
Say ' Diag_expected=' myDiag.DIAG_EXPECTED
Say ' Diag_commerr=' myDiag.DIAG_COMMERR
Say ' Diag_text=' myDiag.DIAG_TEXT
End
End

```

HWILIST — Retrieve HMC and BCPii configuration-related information

Call the HWILIST service to retrieve hardware management console (HMC) and BCPii configuration-related information. Depending on which information is requested, the data returned by this service can be used on subsequent BCPii service calls to take the following actions:

- Connect to a central processor complex (CPC), image (LPAR), capacity record (CAPREC), reset activation profile, image activation profile, or load activation profile using the HWICONN API.
- Register for the proper events (HWIEVENT) using the HWIEVENT API.
- Connect to the local CPC or image.
- Connect to a user-defined image group.

Note: A returned CPC name does not guarantee that an application will be able to connect to that particular resource using the HWICONN API. Connecting to a CPC involves setup issues such as setting up connectivity to a support element and defining the necessary BCPii community name on both the support element and the security product. For more information about the steps that need to be completed before connectivity to a particular CPC is complete, see “Setting up connectivity to the support element” on page 258 and “Community name defined in the security product for each CPC” on page 263.

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
Dispatchable unit mode:	Task
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:	Control parameters must be in the primary address space and addressable by the caller
Linkage:	Standard MVS linkage conventions are used

Programming requirements

See “Syntax, linkage and programming considerations” on page 271 for details about how to call BCPii services in the various programming languages.

REXX programming considerations for the HWILIST service

All information for the HWILIST service applies for REXX requests except:

- An answer area stem variable (for example, AnswerArea) replaces AnswerArea_Ptr.

HWILIST

- AnswerArea.0 replaces NumOfDataItemsReturned.
- AnswerArea.*i* will contain the *i*-th list value on return. For a list type of HWI_LIST_EVENTS, AnswerArea.*i* will contain the *i*-th event bit value on return.
- AnswerAreaLen is not returned.

Restrictions

BCPii does not allow HWILIST to be issued from within a BCPii ENF exit routine.

BCPii does not allow HWILIST with a ListType of HWI_LIST_EVENTS to be issued by a REXX exec running in the TSO/E REXX or ISV-provided REXX environments.

Authorization

The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

For a ListType of HWI_LIST_CPCS, when BCPii is creating the list of CPC network addresses, only those CPC network addresses that the application has at least read access to are listed. The HWI.TARGET.*netid.nau* FACILITY class resource is consulted to determine this.

For a ListType of HWI_LIST_IMAGES, when BCPii is creating the list of image (LPAR) names, only those image names that the application has at least read access to are listed. The HWI.TARGET.*netid.nau.imagename* FACILITY class resource is consulted to determine this.

For a ListType of HWI_LIST_CAPRECS, when BCPii is creating the list of capacity records, only those capacity records that the application has at least read access to are listed. The HWI.CAPREC.*netid.nau.caprecid* FACILITY class resource is consulted to determine this.

For a ListType of HWI_LIST_EVENTS, an application must have at least read access to the SAF-protected FACILITY class resource HWI.TARGET.*netid.nau* for a CPC connection; or at least read access to the SAF-protected FACILITY class resource HWI.TARGET.*netid.nau.imagename* for an image connection.

For a ListType of HWI_LIST_LOCALCPC, an application must have at least read access to the HWI.TARGET.*netid.nau* FACILITY class resource profile where *netid.nau* represents the local CPC network address.

For a ListType of HWI_LIST_LOCALIMAGE, an application must have at least read access to the HWI.TARGET.*netid.nau.imagename* FACILITY class resource profile where *netid.nau* represents the local CPC network address and *imagename* represents the local image (LPAR) name.

For a ListType of HWI_LIST_RESET_ACTPROF, HWI_LIST_IMAGE_ACTPROF, or HWI_LIST_LOAD_ACTPROF, when BCPii is creating the list of activation profiles names, an application needs to have at least read access to the HWI.TARGET.*netid.nau* FACILITY class resource for the CPC to which the activation profiles apply.

For a ListType of HWI_LIST_IMAGEGROUPS, an application must have at least read access to the HWI.TARGET.*netid.nau* FACILITY class resource for the CPC on which image groups may be defined.

Note: BCPii requires the FACILITY class to be RACLIST-specified.

Syntax

Write the call as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Non-REXX parameters	REXX parameters
CALL HWILIST(ReturnCode, ConnectToken, ListType, NumOfDataItemsReturned, AnswerArea_Ptr, AnswerAreaLen, DiagArea);	address bcpii "hwilist ReturnCode ConnectToken ListType AnswerArea. DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

ReturnCode contains the return code from the service.

ConnectToken

Supplied parameter

- Type: Character string
- Length: 16 bytes

ConnectToken represents a logical connection between the application and a CPC, image, or other entity. The ConnectToken is an output parameter on the HWICONN service call.

In most cases, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call. For BCPii REXX execs running under TSO/E or ISV-provided REXX environments, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same task.

If the ListType is HWI_LIST_CPCS, HWI_LIST_LOCALCPC, or HWI_LIST_LOCALIMAGE, this parameter is not relevant and is ignored.

If the ListType is HWI_LIST_IMAGES, this request must either be directed to a specific CPC or to a specific user-defined image group. Therefore, a connect token that represents an already active HWI CPC connection or user-defined image group must be specified.

If the ListType is HWI_LIST_CAPRECS, any of the activation profile (APROF) list types, or HWI_LIST_IMAGEGROUPS, this request must be directed to a specific CPC. Therefore, a connect token that represents an already active HWI CPC connection must be specified.

For a ListType of HWI_LIST_EVENTS, the connect token must represent an already active HWI CPC or image connection, depending on which events are to be listed. If a list of CPC events is required, the connect token must represent an active CPC connection. Likewise, if a list of image events is required, the connect token must represent an active image connection.

HWILIST

ListType

Supplied parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

ListType specifies the type of request for the service.

Constant in Hexadecimal (Decimal) Equate Symbol	Description
1 (1) HWI_LIST_CPCS	Requests a list of CPCs that can be accessed.
2 (2) HWI_LIST_IMAGES	Requests a list of image names that can be accessed on the CPC or within the user-defined image group specified.
3 (3) HWI_LIST_EVENTS	Requests a list of previously subscribed events. Note: This ListType is not supported for REXX execs running in the TSO/E or ISV-provided REXX environments.
4 (4) HWI_LIST_CAPRECS	Requests a list of capacity record ID names that can be accessed.
5 (5) HWI_LIST_LOCALCPC	Requests the name of the local CPC on which the caller is currently executing.
6 (6) HWI_LIST_LOCALIMAGE	Requests the name of the local image (LPAR) on which the HWILIST caller is currently executing.
7 (7) HWI_LIST_RESET_ACTPROF	Requests a list of the currently defined reset activation profiles.
8 (8) HWI_LIST_IMAGE_ACTPROF	Requests a list of the currently defined image activation profiles.
9 (9) HWI_LIST_LOAD_ACTPROF	Requests a list of the currently defined load activation profiles.
A (10) HWI_LIST_IMAGEGROUPS	Requests a list of the currently defined user-defined image groups. Note: This ListType is only available when targeting a z10 or higher CPC.

NumofDataItemsReturned (non-REXX)

Returned parameter

- Type: Integer

- Length: 4 bytes

NumofDataItemsReturned contains the number of data items returned in the answer area.

AnswerArea_Ptr (non-REXX)

AnswerArea. (REXX)

Supplied parameter

- Type: Pointer (non-REXX), stem variable (REXX)
- Length: 4 bytes (non-REXX)

Non-REXX:

AnswerArea_Ptr specifies the address of the answer area where the requested data is returned.

REXX:

A list of the requested objects is returned in an array form of *x.n*; where *x* is the user-defined AnswerArea stem variable and *n* is the n-th element in the stem array.

The AnswerArea.0 stem variable counter holds the number of items returned.

The ListType specified determines the format of the returned data.

ListType	Data to be returned (non-REXX)	Data to be returned (REXX)
HWI_LIST_CPCS	A string comprised of a list of blank-separated concatenated 17-character CPC network addresses. Each network address is in the form of a 1- through 8-character netid, followed by a period, and followed by a 1- through 8-character network addressable unit (NAU) name. The network address is padded with trailing blanks if the total string length of the network address is less than 17 characters. Example: net1.cpc01.	A stem array list of CPC network addresses. Each network address is in the form of a 1- through 8-character netid, followed by a period, and followed by a 1- through 8-character network addressable unit (NAU) name. Example: net1.cpc01.
HWI_LIST_IMAGES	A string comprised of a list of blank-separated concatenated 8-character image names padded with trailing blanks.	A stem array list of image names.
HWI_LIST_EVENTS	A 128-bit string. The first 96 bits (12 bytes) is an eye-catcher value of HWIEVENTBLCK. The last 32 bits represents events already registered for notification. These events were registered by previous HWIEVENT ADD service calls. The returned event indicators are specific to the ConnectToken specified. These indicators are mapped by the type structure HWI_EVENTIDS_TYPE from the BCPii services interface declaration file. If a particular indicator is on, that event is active for this connection.	A stem array list of Boolean values of the EventIDs, which are represented by the EventIDs tail labels defined in HWICIREX. For example, if x is the answerarea stem variable, the returned Boolean data indicates the event registration status. x.Hwi_Event_CmdResp = 1 (on) x.Hwi_Event_StatusChg = 0 (off) : : : x.Hwi_Event_PowerChange = 0 (off) Note: This ListType is not supported for REXX execs running in the TSO/E or ISV-provided REXX environments.
HWI_LIST_CAPRECS	A string comprised of a list of blank-separated concatenated 8-character CAPREC names padded with trailing blanks.	A stem array list of CAPREC names.

HWILIST

ListType	Data to be returned (non-REXX)	Data to be returned (REXX)
HWI_LIST_LOCALCPC	A 17-character string representing the CPC network address of the local CPC. The network address is in the form of a 1- to 8-character netid, followed by a period, followed by a 1- to 8-character network addressable unit (NAU) name. The network address is padded with trailing blanks.	The CPC network address of the local CPC is returned in the first and only element in the stem array. The network address is in the form of a 1-through 8-character netid, followed by a period, and followed by a 1- through 8-character network addressable unit (NAU) name.
HWI_LIST_LOCALIMAGE	An 8-character string representing the image name of the local image (LPAR) padded with trailing blanks.	The image name of the local image (LPAR) is returned in the first and only element in the stem array.
HWI_LIST_RESET_ACTPROF	A string comprised of a list of concatenated 16-character reset activation profile names padded with trailing blanks.	A stem array list of reset activation profile names.
HWI_LIST_IMAGE_ACTPROF	A string comprised of a list of concatenated 16-character image activation profile names padded with trailing blanks.	A stem array list of image activation profile names.
HWI_LIST_LOAD_ACTPROF	A string comprised of a list of concatenated 16-character load activation profile names padded with trailing blanks.	A stem array list of load activation profile names.
HWI_LIST_IMAGEGROUPS	A null-terminated string of null-separated user-defined image group names.	A stem array list of user-defined image group names.

AnswerAreaLen (non-REXX)

Supplied parameter

- Type: Integer
- Length: 4 bytes

AnswerAreaLen specifies the length in bytes of the AnswerArea pointed to by the AnswerArea_Ptr. The amount of storage required by the application at the AnswerArea_Ptr location depends primarily on two factors:

1. The ListType specified
2. The number of data items expected to be returned

For example, if a ListType of HWI_LIST_CPCS is specified and the current HMC LAN has 7 CPCs connected to it, at least 17 bytes x 7 CPCs + the number of blank spaces among the CPCs = 119 + 6 = 125 bytes of data are required for the AnswerArea.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the DiagArea might not be filled in, and the data returned in the area should be ignored.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code that is returned from the console application API or the BCPii transport layer.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, “BCPii communication error reason codes,” on page 583 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0005yyyy' because of one of the following reasons:

Table 65. Reasons for abend X'042', RC X'0005yyyy'

yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See *z/OS MVS System Codes* for additional information.

Return codes

When the service returns control to the caller, GPR 15 and ReturnCode contain a hexadecimal return code.

Return Code in Hexadecimal Equate Symbol	Meaning and Action
0 HWI_OK	Meaning: Successful completion. Action: None.

HWILIST

Return Code in Hexadecimal Equate Symbol	Meaning and Action
100 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified connect token is not valid. This return code indicates one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The connect token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken. • The connect token does not represent an active connection. The connection specified might have already been disconnected using the HWIDISC service call. • The connect token is not associated with the address space of the caller. The ConnectToken specified is associated with a different address space than the caller of this service call. <p>Action: Check for probable coding error.</p>
101 HWI_COMMUNICATION_ERROR	<p>Meaning: A communication error is detected. The hardware management console application API (HWMCA) or the BCPii transport layer has returned with a failing return code.</p> <p>Action: See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPii transport layer.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, "BCPii communication error reason codes," on page 583.</p>
102 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify the specified DiagArea is defined as a 32-byte character field.</p>
103 HWI_CONNECT_TOKEN_INACTIVE	<p>Meaning: The specified connect token is no longer valid. The connection has been disconnected, or it is in the progress of being disconnected.</p> <p>Action: Check for probable coding error. Verify that the specified connect token is still active. If connectivity to the targeted CPC connection no longer exists, all connections associated with that CPC will no longer have a connect token that can be used.</p>

Return Code in Hexadecimal Equate Symbol	Meaning and Action
301 HWI_LISTTYPE_INV	<p>Meaning: Program error. The requested LISTTYPE specified in the call is not valid. The system rejects the service call. This return code indicates one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The ListType specified is not in the acceptable value range of possible list types. • The ListType specified is incompatible with the InConnectToken specified. For example: <ul style="list-style-type: none"> – The ListType specified applies only to CPC connections, but the ConnectToken specified represents an image connection. – The ListType specified applies only to image connections, but the ConnectToken specified represents a CPC connection. • For ListType HWI_LIST_EVENTS, the ConnectToken must not represent a capacity record because capacity record events do not have events directly associated with capacity records connections. Capacity-related events are associated with a CPC connection. <p>Action: Check for probable coding error. Validate that the ListType specified is in the valid range of possible values, and that the ListType specified is permitted for the specified connection type.</p>
302 HWI_DATA_EXCEEDED	<p>Meaning: Program error. The amount of returned data exceeded the size of the answer area. No data or only partial data is returned.</p> <p>Action: Check for probable coding error. See the DiagArea for further diagnostic information. The Diag_Actual indicates the application-specified length. The Diag_Expected indicates the size required for the AnswerArea.</p>
303 HWI_ANSWERAREA_INACCESSIBLE	<p>Meaning: Program error. The answer area data area is either partially or completely inaccessible by the application and the Base Control Program internal interface (BCPii) address space.</p> <p>Action: Check for probable coding error. Verify that the AnswerArea_Ptr points to a data area where the answer area is and make sure the data area is accessible.</p>

HWILIST

Return Code in Hexadecimal Equate Symbol	Meaning and Action
304 HWI_LIST_NODATA_RETURNED	<p>Meaning: There is no data to be returned or the caller does not have enough access to display the listed values.</p> <p>Action: Check for probable coding error. Verify that proper access is granted for the request.</p>
305 HWI_LISTTYPE_NOT_SUPPORTED	<p>Meaning: The targeted hardware of the HWILIST request does not support the request attempted by the program.</p> <p>Action: Verify that the targeted hardware supports the type of request being made.</p>
F00 HWI_NOT_AVAILABLE	<p>Meaning: BCPii services are not available, and the system rejects the service request.</p> <p>Action: Notify the system programmer to start the BCPii address space and try the request again. See “Restarting the HWIBCPii address space” on page 269 about how to start the BCPii address space.</p> <p>Programs can also listen to ENF68 to determine when BCPii services are available. See <i>z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG</i> for how to listen for BCPii activation messages.</p>
F01 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.</p> <p>Action: Check the calling program for a probable coding error.</p>

Return Code in Hexadecimal Equate Symbol	Meaning and Action
F02 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV. • Define read access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau.imagename</i> for HWI_LIST_IMAGES ListType. • Define read access authorization to the FACILITY class resource profile HWI.CAPREC.<i>netid.nau.caprec</i> for HWI_LIST_CAPRECS ListType. • For a ListType of HWI_LIST_EVENTS, define read access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau</i> for a CPC connection, and HWI.TARGET.<i>netid.nau.imagename</i> for an image connection. • For a ListType of HWI_LIST_LOCALCPC, define read access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau</i> where <i>netid.nau</i> represents the local CPC network address. • For a ListType of HWI_LIST_LOCALIMAGE, define read access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau.imagename</i> where <i>netid.nau</i> represents the local CPC network address and <i>imagename</i> represents the local image (LPAR) name. • For the ListType of HWI_LIST_RESET_ACTPROF, HWI_LIST_IMAGE_ACTPROF, HWI_LIST_LOAD_ACTPROF, or HWI_LIST_IMAGEGROUPS, define read access authorization to the FACILITY class resource profile HWI.TARGET.<i>netid.nau</i> for the CPC where the activation profiles or image groups to be listed are defined. • Ensure that the referenced facility class profiles are RACLIST-specified.
F03 HWI_INTERRUPT_STATUS_INV	<p>Meaning: The calling program is disabled. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

HWILIST

Return Code in Hexadecimal Equate Symbol	Meaning and Action
F04 HWI_MODE_INV	<p>Meaning: The calling program is not in task mode. The system rejects this service request.</p> <p>Action: Check the calling program for a probable error.</p>
F05 HWI_LOCKS_HELD	<p>Meaning: The calling program is holding one or more locks. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F06 HWI_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects this service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again.</p>
F07 HWI_UNSUPPORTED_ENVIRONMENT	<p>Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine).</p> <p>Action: Issue the BCPii service from a different execution environment.</p>
FFF HWI_UNEXPECTED_ERROR	<p>Meaning: System error. The service that was called encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Example

In the pseudocode example, the caller issues a call to retrieve a list CPCs that can be accessed.

```

.
.
ListType = HWI_LIST_CPCS;
AnswerArea_Ptr = addr(AnswerArea);
AnswerAreaLen = 125;
CALL HWILIST (ReturnCode, ConnectToken, ListType, NumofDataItemsReturned,
              AnswerArea_Ptr, AnswerAreaLen, DiagArea)
.
.

```

A REXX programming example for the HWILIST service:

```

myListType = HWI_LIST_IMAGES

address bcpii
    "hwilist RetCode myConnectToken myListType myAnswerArea. myDiag."

```

```

If (RC <> 0) | (Retcode <> 0) Then

```

```

Do
  Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
  If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
    Do
      Say ' Diag_index=' myDiag.DIAG_INDEX
      Say ' Diag_key=' myDiag.DIAG_KEY
      Say ' Diag_actual=' myDiag.DIAG_ACTUAL
      Say ' Diag_expected=' myDiag.DIAG_EXPECTED
      Say ' Diag_commerr=' myDiag.DIAG_COMMERR
      Say ' Diag_text=' myDiag.DIAG_TEXT
    End
  Else
    Do
      Say 'Number of items returned = 'myAnswerArea.0 /* Count of items returned */

      If myAnswerArea.0 > 0 Then
        Do n=1 to myAnswerArea.0
          Say 'Image #'n' = 'myAnswerArea.n
        End
      End
    End
  End

```

HWIQUERY — BCPii retrieval of SE/HMC-managed attributes

Call the HWIQUERY service to retrieve information about objects managed by the support element (SE) or hardware management console (HMC) related with central processor complexes (CPCs), CPC images (LPARs), capacity records, different types of activation profiles, or user-defined image groups.

For some connection types (HWI_CPC and HWI_IMAGE in particular), grouping multiple attributes together into a single HWIQUERY service call may result in significantly reduced waiting times rather than querying the same number of attributes one at a time. Whenever possible, an application should consolidate its HWIQUERY service calls to query multiple attributes using the same query request.

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
Dispatchable unit mode:	Task
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:	Control parameters must be in the primary address space and addressable by the caller
Linkage:	Standard MVS linkage conventions are used

Programming requirements

See “Syntax, linkage and programming considerations” on page 271 for details about how to call BCPii services in the various programming languages.

See “HWIQUERY and HWISET” on page 585 for the summary table of the BCPii HWIQUERY and HWISET attributes and the objects that can be targeted for each function.

REXX programming considerations for the HWIQUERY service

All information for the HWIQUERY service applies for REXX requests except:

- A query parameter stem variable (for example, QueryParm) replaces QueryParm_Ptr.
 - QueryParm.0 replaces NumOfAttributes. QueryParm.0 is required to specify the number of attributes to be queried. The maximum number of attributes allowed is 64.
 - QueryParm.n.ATTRIBUTEIDENTIFIER must contain the *n*-th attribute identifier to be returned.
 - QueryParm.n.ATTRIBUTEVALUE will contain the *n*-th attribute value on return.
- AttributeValue_Ptr is replaced with AttributeValue.
- AttributeValueLen is not used.
- AttributeValueLenReturned is not used.
- For the PSW (HWI_PSW) attribute:
 - QueryParm.n.ATTRIBUTEVALUE.0 will contain the number of PSWs returned (*j*).
 - QueryParm.n.ATTRIBUTEVALUE.m.CPUID will contain the *m*-th CPU identifier.
 - QueryParm.n.ATTRIBUTEVALUE.m.PSW will contain the *m*-th PSW.
- For the supported processor power savings mode (HWI_SUPPPPOWERMODE) attribute:
 - QueryParm.n.ATTRIBUTEVALUE.0 will contain the number of supported power savings modes returned (*m*).
 - QueryParm.n.ATTRIBUTEVALUE.m.PSMODE will contain the *m*-th supported power savings mode.
- For the list of IP addresses (HWI_LIST_IP_ADDRESSES) attribute:
 - QueryParm.n.ATTRIBUTEVALUE.0 will contain the number of IP addresses returned (*j*).
 - QueryParm.n.ATTRIBUTEVALUE.m.IPADDR will contain the *m*-th IP address.

Restrictions

BCPii does not allow HWIQUERY to be issued from within a BCPii ENF exit routine.

Authorization

The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

Client application must have at least read access to the SAF-protected FACILITY class HWI.TARGET.netid.nau for any CPC, activation profile, or user-defined image group queries, or HWI.TARGET.netid.nau.imagename for image queries, or HWI.CAPREC.netid.nau.caprecid for capacity record queries.

Note: BCPii requires the FACILITY class to be RACLIST-specified.

Syntax

Write the call as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Non-REXX parameters	REXX parameters
CALL HWIQUERY(ReturnCode, ConnectToken, QueryParm_Ptr, NumOfAttributes, DiagArea);	address bcpil "hwiquery ReturnCode ConnectToken QueryParm. DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

ReturnCode contains the return code from the service.

ConnectToken

Supplied parameter

- Type: Character string
- Length: 16 bytes

ConnectToken represents a logical connection between the application and a CPC, image, capacity record, activation profile, or user-defined image group. The ConnectToken is an output parameter on the HWICONN service call.

In most cases, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call. For BCPii REXX execs running under the TSO/E or ISV-provided REXX environments, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same task.

QueryParm_Ptr (non-REXX)

QueryParm. (REXX)

Supplied parameter

- Type: Pointer (non-REXX), stem variable (REXX)
- Length: 4 bytes (non-REXX)

Non-REXX:

QueryParm_Ptr specifies the address of a user-defined query structure that contains a list of one or more requested attributes to be queried, in the following form: attribute that is required, address of where returned value is to be stored, the length of the storage available to HWIQUERY to store the returned value, and the actual length of the data that will be returned in the data area.

The size of the data area pointed to by this parameter must be 16 bytes multiplied by the NumOfAttributes parameter. For example, if NumofAttributes is 4, the data area pointed to by this parameter must be at least 64 bytes long (16 x 4).

The storage area that contains each attribute in the QueryParm is shown in the following table:

HWIQUERY

Field name	Field type
AttributeIdentifier	32-bit unsigned integer
AttributeValue_Ptr	Pointer
AttributeValueLen	32-bit unsigned integer
AttributeValueLenReturned	32-bit unsigned integer

This table is mapped by the data structure `Hwi_QueryParm_Type` in the data mappings provided for the various programming languages supported. See “Syntax, linkage and programming considerations” on page 271 for more information.

If all of the data can be written into the data area (the `AttributeValueLen` is greater than or equal to the actual data returned), the `AttributeValueLenReturned` field contains the actual length of the data written in the storage specified at address `AttributeValue_Ptr`.

The `AttributeValueLenReturned` is only used as an output parameter. Any value contained in the field when `HWIQUERY` is called is ignored.

REXX:

`QueryParm` is a compound (stem) variable which contains one or more requested attributes to be queried and returned.

The compound (stem) variable is specified as follows (where x is the user-defined `QueryParm` stem variable and n is the n -th attribute for the request):

- **`x.0` specifies the number of attributes to be queried.** The maximum number of attributes allowed is 64. (Supplied parameter)
- `x.n.ATTRIBUTEIDENTIFIER` specifies the requested attribute. Set this variable to one of the query attribute constants defined in `HWICIREX`. (Supplied parameter)
- `x.n.ATTRIBUTEVALUE` is the data value to be returned for most attributes. (Returned parameter)
- Some single attributes can return multiple objects in a formatted structure. For those attributes, `x.n.ATTRIBUTEVALUE.0` (Returned parameter) is the total number of returned objects. See the following query attribute table for the following attributes that are in a different format. These attributes include: `HWI_SUPPPPOWERMODE`, `HWI_LIST_IP_ADDRESSES`, and `HWI_PSWS`.

The following table lists the valid query attribute identifiers. For more information about these attributes, see the following publications:

- *System z Application Programming Interfaces (SB10-7030-13)*
- *System z10 and eServer zSeries Application Programming Interfaces (SB10-7030-09)*
- *System z9 and eServer zSeries Application Programming Interfaces (SB10-7030-08)*
- Publication appropriate to the level of hardware that the `HWIQUERY` is targeted

Constant in hexadecimal (Decimal) Equate symbol	Description																
1 (1) HWI_NAME	Requests to retrieve the name that represents the connect token. Note: The input connection token must represent a <i>CPC connection</i> , an <i>image connection</i> , a <i>reset activation profile connection</i> , an <i>image activation profile connection</i> , a <i>load activation profile connection</i> , or an <i>image group connection</i> .																
2 (2) HWI_ERRSTAT	Requests to retrieve whether the status is acceptable. Note: The input connection token must represent a <i>CPC connection</i> an <i>image connection</i> , or an <i>image group connection</i> .																
3 (3) HWI_BUSYSTAT	Requests to retrieve whether the status is busy. Note: The input connection token must represent a <i>CPC connection</i> an <i>image connection</i> , or an <i>image group connection</i> .																
4 (4) HWI_MSGSTAT	Requests to retrieve whether hardware messages are present. Note: The input connection token must represent a <i>CPC connection</i> or an <i>image connection</i> .																
5 (5) HWI_OPERSTAT	Requests to retrieve the current status. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .																
6 (6) HWI_ACCSTAT	Requests to retrieve the acceptable status values. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .																
7 (7) HWI_APROF	Requests to retrieve the next activation reset profile name. Note: The input connection token must represent a <i>CPC connection</i> or an <i>image connection</i> .																
8 (8) HWI_LUAPROF	Requests to retrieve the last used activation profile. Note: The input connection token must represent a <i>CPC connection</i> or an <i>image connection</i> .																
9 (9) HWI_OBJTYPE	Requests to retrieve the object type. <table border="0"> <tr> <td>Input connection token represents</td> <td>Returns</td> </tr> <tr> <td>CPC</td> <td>HMCA_CPC_OBJECT</td> </tr> <tr> <td>CPC image</td> <td>HMCA_CPC_IMAGE_OBJECT</td> </tr> <tr> <td>Capacity record</td> <td>HMCA_CAPACITY_RECORD</td> </tr> <tr> <td>Reset activation profile</td> <td>HMCA_ACT_PROFILE_RESET</td> </tr> <tr> <td>Image activation profile</td> <td>HMCA_ACT_PROFILE_IMAGE</td> </tr> <tr> <td>Load activation profile</td> <td>HMCA_ACT_PROFILE_LOAD</td> </tr> <tr> <td>Image Group</td> <td>HMCA_CPC_IMAGE_USER_GROUP</td> </tr> </table> Note: The input connection token must represent a <i>CPC connection</i> , an <i>image connection</i> , a <i>capacity record connection</i> , a <i>reset activation profile connection</i> , an <i>image activation profile connection</i> , a <i>load activation profile connection</i> , or an <i>image group connection</i> .	Input connection token represents	Returns	CPC	HMCA_CPC_OBJECT	CPC image	HMCA_CPC_IMAGE_OBJECT	Capacity record	HMCA_CAPACITY_RECORD	Reset activation profile	HMCA_ACT_PROFILE_RESET	Image activation profile	HMCA_ACT_PROFILE_IMAGE	Load activation profile	HMCA_ACT_PROFILE_LOAD	Image Group	HMCA_CPC_IMAGE_USER_GROUP
Input connection token represents	Returns																
CPC	HMCA_CPC_OBJECT																
CPC image	HMCA_CPC_IMAGE_OBJECT																
Capacity record	HMCA_CAPACITY_RECORD																
Reset activation profile	HMCA_ACT_PROFILE_RESET																
Image activation profile	HMCA_ACT_PROFILE_IMAGE																
Load activation profile	HMCA_ACT_PROFILE_LOAD																
Image Group	HMCA_CPC_IMAGE_USER_GROUP																
A (10) HWI_IMLMODE	Requests to retrieve the initial machine load (IML) mode (LPAR). Note: The input connection token must only represent a <i>CPC connection</i> or an <i>image connection</i> .																

HWIQUERY

Constant in hexadecimal (Decimal) Equate symbol	Description
B-16 (11-22) RESERVED	Reserved for attributes that are common to CPC and image connections unless otherwise noted.
17 (23) HWI_IPADDR	Requests to retrieve the internet address (IPv4 format). Note: The input connection token must only represent a <i>CPC connection</i> .
18 (24) HWI_SNAADDR	Requests to retrieve the SNA address (<i>netid.nau</i>). Note: The input connection token must only represent a <i>CPC connection</i> .
19 (25) HWI_MMODEL	Requests to retrieve the machine model. Note: The input connection token must only represent a <i>CPC connection</i> .
1A (26) HWI_MTYPE	Requests to retrieve the machine type. Note: The input connection token must only represent a <i>CPC connection</i> .
1B (27) HWI_MSERIAL	Requests to retrieve the machine serial. Note: The input connection token must only represent a <i>CPC connection</i> .
1C (28) HWI_CPCSERIAL	Requests to retrieve the CPC serial number. Note: The input connection token must only represent a <i>CPC connection</i> .
1D (29) HWI_CPCID	Requests to retrieve the CPC identifier. Note: The input connection token must only represent a <i>CPC connection</i> .
1E (30) HWI_RESERVEID	Requests to retrieve the name of the application that is holding the reserve (if any). Note: The input connection token must only represent a <i>CPC connection</i> .
1F (31) HWI_SVCEREQD	Requests to retrieve the service required. Note: The input connection token must only represent a <i>CPC connection</i> .
20 (32) HWI_CBUINSTD	Requests to retrieve the CBU installed. Note: The input connection token must only represent a <i>CPC connection</i> .
21 (33) HWI_CBUENABLD	Requests to retrieve the CBU enabled. Note: The input connection token must only represent a <i>CPC connection</i> .

Constant in hexadecimal (Decimal) Equate symbol	Description
22 (34) HWI_CBUACTIVE	Requests to retrieve the CBU activated. Note: The input connection token must only represent a <i>CPC connection</i> .
23 (35) HWI_CBUACTDT	Requests to retrieve the CBU activation date. Note: The input connection token must only represent a <i>CPC connection</i> .
24 (36) HWI_CBUEXPDT	Requests to retrieve the CBU expiration date. Note: The input connection token must only represent a <i>CPC connection</i> .
25 (37) HWI_CBUTESTAR	Requests to retrieve the CBU tests left (test activations remaining). Note: The input connection token must only represent a <i>CPC connection</i> .
26 (38) HWI_CBUREALAV	Requests to retrieve the CBU real activation available. Note: The input connection token must only represent a <i>CPC connection</i> .
27 (39) HWI_PRUNTYPE	Requests to retrieve the processor running time type. Note: The input connection token must only represent a <i>CPC connection</i> or a <i>reset activation profile connection</i> .
28 (40) HWI_PRUNTIME	Requests to retrieve the processor running time. Note: The input connection token must only represent a <i>CPC connection</i> or a <i>reset activation profile connection</i> .
29 (41) HWI_PRUNTSEW	Requests to retrieve the processor running time slice end wait processing. Note: The input connection token must only represent a <i>CPC connection</i> or a <i>reset activation profile connection</i> .
2A (42) HWI_OOCINST	Requests to retrieve the on and off capacity on demand installed. Note: The input connection token must only represent a <i>CPC connection</i> .
2B (43) HWI_OOCACT	Requests to retrieve the on and off capacity on demand currently activated. Note: The input connection token must only represent a <i>CPC connection</i> .
2C (44) HWI_OOCENAB	Requests to retrieve the on and off capacity on demand enabled. Note: The input connection token must only represent a <i>CPC connection</i> .
2D (45) HWI_OOCADT	Requests to retrieve the on and off capacity on demand activation date. Note: The input connection token must only represent a <i>CPC connection</i> .

HWIQUERY

Constant in hexadecimal (Decimal) Equate symbol	Description
2E (46) HWI_PCPCSWM	Requests to retrieve the permanent CPC software model. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
2F (47) HWI_PPBPMSW	Requests to retrieve the permanent plus billable processor software model. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
30 (48) HWI_PPTPSW	Requests to retrieve the permanent plus (all) temporary processor software model. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
31 (49) HWI_PCPCMSU	Requests to retrieve the permanent CPC millions of service units (MSU) value. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
32 (50) HWI_PPBPMSU	Requests to retrieve the permanent plus billable processor MSU value. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
33 (51) HWI_PPTPMSU	Requests to retrieve the permanent plus (all) temporary processor MSU value. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
34 (52) HWI_NUMGPP	Requests to retrieve the number of general purpose processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
35 (53) HWI_NUMSAP	Requests to retrieve the number of service assist processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
36 (54) HWI_NUMIFAP	Requests to retrieve the number of the integrated facility for applications (IFA) processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
37 (55) HWI_NUMIFLP	Requests to retrieve the number of the integrated facility for Linux (IFL) processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
38 (56) HWI_NUMICFP	Requests to retrieve the number of the internal coupling facility (ICF) processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .
39 (57) HWI_NUMIIPP	Requests to retrieve the number of integrated information processors (IIP). This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .

Constant in hexadecimal (Decimal) Equate symbol	Description						
3A (58) HWI_NUMFLTYP	Requests to retrieve the number of defective (faulty) processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .						
3B (59) HWI_NUMSPARE	Requests to retrieve the number of spare processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .						
3C (60) HWI_NUMPENDP	Requests to retrieve the number of pending (activation) processors. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .						
3D (61) HWI_CAPCHGALLWD	Requests to determine if activate/deactivate of capacity are permitted. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .						
3E (62) HWI_DGRSTAT	Requests to retrieve degraded status. Note: The input connection token must only represent a <i>CPC connection</i> .						
3F (63) HWI_CURRPOWERMODE	Requests to retrieve the current processor power savings mode active on the targeted CPC. This attribute is only available when targeting a zEnterprise or higher CPC. Note: The input connection token must only represent a <i>CPC connection</i> .						
40 (64) HWI_SUPPPPOWERMODE	<p>Requests to retrieve the supported processor power savings modes available on the targeted CPC. This attribute is only available when targeting a zEnterprise or higher CPC.</p> <p>Non-REXX: The returned data is mapped as follows:</p> <table border="0" data-bbox="678 1266 998 1402"> <thead> <tr> <th style="text-align: left;">Field Name</th> <th style="text-align: left;">Field Type</th> </tr> <tr> <th style="text-align: left;">-----</th> <th style="text-align: left;">-----</th> </tr> </thead> <tbody> <tr> <td style="text-align: left;">Number of supported powersave modes</td> <td style="text-align: left;">32-bit integer</td> </tr> </tbody> </table> <p>For each supported powersave mode, the following is returned:</p> <p>Powersave mode 32-bit integer value Note: The query parameter for this attribute must specify a data area large enough to contain all of the structure (that is, 32 bits + 32 bits per supported powersave mode returned). For example, if there are 2 supported powersave modes on the targeted CPC, then the structure must be at least 32 + (32 × 2) = 96 bits (12 bytes).</p> <p>REXX: The returned data is mapped as follows (where <i>x</i> is the user-defined QueryParm stem, <i>n</i> is the <i>n</i>-th requested attribute and <i>m</i> is the <i>m</i>-th returned powersave mode value):</p> <ul style="list-style-type: none"> • <i>x.n.ATTRIBUTEVALUE.0</i> is the number of supported powersave modes (<i>m</i>). • <i>x.n.ATTRIBUTEVALUE.m.PSMODE</i> is the <i>m</i>-th powersave mode value. <p>Note: The input connection token must only represent a <i>CPC connection</i>.</p>	Field Name	Field Type	-----	-----	Number of supported powersave modes	32-bit integer
Field Name	Field Type						
-----	-----						
Number of supported powersave modes	32-bit integer						

HWIQUERY

Constant in hexadecimal (Decimal) Equate symbol	Description
41 (65) HWI_STPCONFIG	Requests to retrieve the Server Timer Protocol (STP) configuration data. Note: The input connect token must only represent a <i>CPC connection</i> .
42 (66) HWI_NUMPGPP	Requests to retrieve the number of pending general purpose processors. Note: The input connect token must only represent a <i>CPC connection</i> .
43 (67) HWI_NUMPSAP	Requests to retrieve the number of pending service assist processors. Note: The input connect token must only represent a <i>CPC connection</i> .
44 (68) HWI_NUMPAAP	Requests to retrieve the number of pending Application Assist Processor (AAP) processors. Note: The input connect token must only represent a <i>CPC connection</i> .
45 (69) HWI_NUMPIFLP	Requests to retrieve the number of pending Integrated Facility for Linux (IFL) processors. Note: The input connect token must only represent a <i>CPC connection</i> .
46 (70) HWI_NUMPICFP	Requests to retrieve the number of pending Internal Coupling Facility (ICF) processors. Note: The input connect token must only represent a <i>CPC connection</i> .
47 (71) HWI_NUMPIIPP	Requests to retrieve the number of pending Integrated Information (IIP) processors. Note: The input connect token must only represent a <i>CPC connection</i> .
48 (72) HWI_POWERMODEALLOWED	Requests to retrieve the processor power savings mode allowed. This attribute is only available when targeting a zEnterprise or higher CPC. HWMCA_TRUE The processor currently allows switching to power savings mode. HWMCA_FALSE The processor currently does not allow switching to power savings mode. Note: The input connection token must only represent a <i>CPC connection</i> .
49 (73) HWI_VERSION	Requests to retrieve the CPC version number. Note: The input connection token must only represent a <i>CPC connection</i> .
4A (74) HWI_EC_MCL_INFO	Requests to retrieve an XML string that describes the Engineering Change (EC) and Microcode Level (MCL) levels. Note: The input connection token must only represent a <i>CPC connection</i> . Attention: The data returned by the support element can be quite large. Consider using a larger data area when requesting this attribute.

Constant in hexadecimal (Decimal) Equate symbol	Description						
<p>4B (75) HWI_LIST_IP_ADDRESSES</p>	<p>Requests to retrieve all the IP addresses (in either IPv4 or IPv6 format, or both) used for the targeted CPC.</p> <p>Non-REXX: The returned data is mapped as follows:</p> <table border="1" data-bbox="678 394 1193 514"> <thead> <tr> <th>Field Name</th> <th>Field Type</th> </tr> </thead> <tbody> <tr> <td>Number of IP addresses</td> <td>32-bit unsigned integer</td> </tr> <tr> <td>IP address value</td> <td>39-character value padded with blanks</td> </tr> </tbody> </table> <p>Note: The query parameter for this attribute must specify a data area large enough to contain all of the structure (that is, a 4-byte length field plus a 39-byte field for each IP address returned). For example, if there are 3 IP addresses returned, the AttributeValueLen specified for this attribute must be at least $(4 + (39 \times 3)) = 121$ bytes.</p> <p>REXX: The returned data is mapped as follows (where <i>x</i> is the user-defined QueryParm stem, <i>n</i> is the <i>n</i>-th requested attribute and <i>m</i> is the <i>m</i>-th returned IP address value):</p> <ul style="list-style-type: none"> • <i>x.n.ATTRIBUTEVALUE.0</i> is the number of IP addresses (<i>m</i>). • <i>x.n.ATTRIBUTEVALUE.m.IPADDR</i> is the <i>m</i>-th IP address value. <p>Note: The input connection token must only represent a <i>CPC connection</i>.</p>	Field Name	Field Type	Number of IP addresses	32-bit unsigned integer	IP address value	39-character value padded with blanks
Field Name	Field Type						
Number of IP addresses	32-bit unsigned integer						
IP address value	39-character value padded with blanks						
<p>4C (76) HWI_AUTO_SWITCH_ENABLED</p>	<p>Requests to retrieve a value used to determine if automatic switching between primary and alternate support elements is enabled.</p> <p>A 4-byte integer type value is returned:</p> <p>HWMCA_TRUE Automatic switching is enabled.</p> <p>HWMCA_FALSE Automatic switching is disabled.</p> <p>Note: The input connection token must only represent a <i>CPC connection</i>.</p>						
<p>4D-68 (77-104) RESERVED</p>	<p>Reserved for CPC attributes unless otherwise noted.</p>						
<p>69 (105) HWI_CPCNAME</p>	<p>Requests to retrieve the parent (CPC) name.</p> <p>Note: The input connection token must only represent an <i>image connection</i>.</p>						

HWIQUERY

Constant in hexadecimal (Decimal) Equate symbol	Description
6A (106) HWI_OSNAME	<p>Requests to retrieve the SW operating system name.</p> <p>The values returned on the HWI_OSNAME attribute are not owned by z/OS BCPii and are subject to change. The possible values returned by the various operating systems at the time of this publication include:</p> <p>HWI_OSTYPE value: MVS The HWI_OSNAME value returned is the SYSNAME parameter as defined in IEASYSxx parmlib member for the targeted image.</p> <p>HWI_OSTYPE value: VM The HWI_OSNAME value returned is the system identifier or system name as defined in the SYSTMID field in the SYSCM (System Common Area) control block.</p> <p>HWI_OSTYPE value: LINUX The HWI_OSNAME value returned is N/A.</p> <p>HWI_OSTYPE value: VSE The HWI_OSNAME value returned is the VSE system name.</p> <p>HWI_OSTYPE value: Z TPF EE The HWI_OSNAME value returned is the id value representing the targeted image's CPU designation in the z/TPF complex. Note: The input connection token must only represent an <i>image connection</i>.</p>
6B (107) HWI_OSTYPE	<p>Requests to retrieve the SW operating system type.</p> <p>The values returned on the HWI_OSTYPE attribute are not owned by z/OS BCPii and are subject to change. Possible values include MVS, VM, LINUX, VSE, and Z TPF EE. Note: The input connection token must only represent an <i>image connection</i>.</p>

Constant in hexadecimal (Decimal) Equate symbol	Description
6C (108) HWI_OSLEVEL	<p>Requests to retrieve the SW operating system level.</p> <p>The values returned on the HWI_OSLEVEL attribute are not owned by z/OS BCPii and are subject to change. The possible values returned by the various operating systems at the time of this publication include:</p> <p>HWI_OSTYPE value: MVS The HWI_OSLEVEL value is mapped by the CVTOSLVL field of the CVT control block.</p> <p>HWI_OSTYPE value: VM The HWI_OSLEVEL value is mapped as follows:</p> <ul style="list-style-type: none"> • 4-bit release # • 4-bit modification level • 8-bit version # • 16-bit service level • 8-bit MVS guest count • 8-bit LINUX guest count • 8-bit VSE guest count • 8-bit Solaris guest count <p>HWI_OSTYPE value: LINUX The HWI_OSLEVEL value is mapped as follows, in hexadecimal:</p> <ul style="list-style-type: none"> • 40 bits N/A • 8-bit major kernel revision • 8-bit major release • 8-bit minor release <p>HWI_OSTYPE value: VSE The HWI_OSLEVEL value is mapped as follows:</p> <ul style="list-style-type: none"> • 32-bit VSE/AF release level • 32-bit latest service level (if available) <p>HWI_OSTYPE value: Z TPF EE The HWI_OSLEVEL value is mapped as follows:</p> <ul style="list-style-type: none"> • 16-bit version # • 8-bit PUT level <p>Examples:</p> <p>For MVS, FFFFFFFFEF7F0000 implies that the target is running z/OS V1R13 because the CVTZOS_V1R13 bit is the last supported release flag that is on.</p> <p>For VM, 4005100200320000 implies that the target is running z/VM Release 4, Modification Level 0, Version 5, Service Level 1002, MVS guest count 0, Linux guest count 32, VSE guest count 0, and Solaris guest count 0.</p> <p>For LINUX, 000000000020620 implies that the target is running z/LINUX major kernel revision 2, major release 6, and minor release 32.</p> <p>For VSE, 0830000000000000 implies that the target is running at the VSE/AF 8.3 release level and no service level is available.</p> <p>For Z TPF EE, 0101070000000000 implies that the target is running z/TPF version 1.1, PUT level 7.</p> <p>Note: The input connection token must represent an <i>image connection</i>.</p>
6D (109) HWI_SYSPLEX	<p>Requests to retrieve the SW sysplex name (z/OS only).</p> <p>Note: The input connection token must only represent an <i>image connection</i>.</p>

HWIQUERY

Constant in hexadecimal (Decimal) Equate symbol	Description
6E (110) HWI_CLUSTER	Requests to retrieve the LPAR cluster name. Note: The input connection token must only represent an <i>image connection</i> .
6F (111) HWI_PARTITIONID	Requests to retrieve the partition ID. If the connection token represents an <i>image connection</i> , the image partition ID is returned; if the connection token represents an <i>image activation profile connection</i> , the image activation profile partition ID is returned. The image partition ID is only retrievable when the partition has been activated. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
70 (112) HWI_DEFCAP	Requests to retrieve the current defined capacity. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
71 (113) HWI_SGPIPW	Requests to retrieve the shared general processor initial processing weight (SGPIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
72 (114) HWI_SGPIPWCAP	Requests to retrieve the SGPIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
73 (115) HWI_SGPPWMIN	Requests to retrieve the minimum SGPPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
74 (116) HWI_SGPPWMAX	Requests to retrieve the maximum SGPPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
75 (117) HWI_SGPPW	Requests to retrieve the current SGPPW value. Note: The input connection token must only represent an <i>image connection</i> .
76 (118) HWI_SGPPWCAP	Requests to retrieve the SGPPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> .
77 (119) HWI_WLM	Requests to retrieve whether WLM is allowed to change processing weight-related attributes. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
78 (120) HWI_IFAIPW	Requests to retrieve the integrated facility for applications initial processing weight (IFAIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
79 (121) HWI_IFAIPWCAP	Requests to retrieve the IFAIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .

Constant in hexadecimal (Decimal) Equate symbol	Description
7A (122) HWI_IFAPWMIN	Requests to retrieve the minimum IFAPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7B (123) HWI_IFAPWMAX	Requests to retrieve the maximum IFAPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7C (124) HWI_IFAPW	Requests to retrieve the current IFAPW value. Note: The input connection token must only represent an <i>image connection</i> .
7D (125) HWI_IFAPWCAP	Requests to retrieve the IFAPW to be currently capped or not capped. Note: The input connection token must only represent an <i>image connection</i> .
7E (126) HWI_IFLIPW	Requests to retrieve the integrated facility for Linux initial processing weight. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7F (127) HWI_IFLIPWCAP	Requests to retrieve the IFLIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
80 (128) HWI_IFLPWMIN	Requests to retrieve the minimum IFLPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
81 (129) HWI_IFLPWMAX	Requests to retrieve the maximum IFLPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
82 (130) HWI_IFLPW	Requests to retrieve current IFLPW value. Note: The input connection token must only represent an <i>image connection</i> .
83 (131) HWI_IFLPWCAP	Requests to retrieve the IFLPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> .
84 (132) HWI_ICFIPW	Requests to retrieve the internal coupling facility initial processing weight (ICFIPW). Note: The input connection token must only represent an <i>image connection</i> (Coupling Facility images only) or an <i>image activation profile connection</i> .
85 (133) HWI_ICFIPWCAP	Requests to retrieve the ICFIPW be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> (Coupling Facility images only) or an <i>image activation profile connection</i> .

HWIQUERY

Constant in hexadecimal (Decimal) Equate symbol	Description
86 (134) HWI_ICFPWMIN	Requests to retrieve the minimum ICFPW value. Note: The input connection token must only represent an <i>image connection</i> (Coupling Facility images only) or an <i>image activation profile connection</i> .
87 (135) HWI_ICFPWMAX	Requests to retrieve the maximum ICFPW value. Note: The input connection token must only represent an <i>image connection</i> (Coupling Facility images only) or an <i>image activation profile connection</i> .
88 (136) HWI_ICFPW	Requests to retrieve the current ICFPW value. Note: The input connection token must only represent an <i>image connection</i> (Coupling Facility images only).
89 (137) HWI_ICFPWCAP	Requests to retrieve the ICFPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> (Coupling Facility images only).
8A (138) HWI_IIPW	Requests to retrieve the integrated information processors initial processing weight (IIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8B (139) HWI_IIPWCAP	Requests to retrieve the IIPW be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8C (140) HWI_IIPWMIN	Requests to retrieve the minimum IIPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8D (141) HWI_IIPWMAX	Requests to retrieve the maximum IIPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8E (142) HWI_IIPW	Requests to retrieve the current IIPW value. Note: The input connection token must only represent an <i>image connection</i> .
8F (143) HWI_IIPWCAP	Requests to retrieve the IIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> .
90 (144) HWI_IPLTOKEN	Requests to retrieve the IPL token associated with the current IPL of the image targeted. Note: The input connection token must only represent an <i>image connection</i> .

Constant in hexadecimal (Decimal) Equate symbol	Description						
91 (145) HWI_PSW	<p>Requests to retrieve the program status word (PSW) for each of the central processors (CP) associated with this image.</p> <p>Non-REXX: The returned data is mapped as follows:</p> <table border="0"> <tr> <td style="padding-right: 20px;">Field Name</td> <td>Field Type</td> </tr> <tr> <td>-----</td> <td>-----</td> </tr> <tr> <td>Number of CPs</td> <td>32-bit unsigned integer</td> </tr> </table> <p>For each CP, the following is returned: CPUID 32-bit unsigned integer PSW 128-bit unsigned integer</p> <p>Note: The query parameter for this attribute must specify a data area large enough to contain all of the above structure (that is 32 bits + 160 bits per CP). For example, if there are 4 CPs on the targeted image, the AttributeValueLen specified for this attribute must be 32 + (160 x 4) = 672 bits (84 bytes).</p> <p>REXX: The returned data is mapped as follows (where <i>x</i> is the user-defined QueryParm stem, <i>n</i> is the <i>n</i>-th requested attribute and <i>m</i> is the <i>m</i>-th returned CPUID or PSW value):</p> <ul style="list-style-type: none"> • <i>x.n.ATTRIBUTEVALUE.0</i> is the number of CPs (<i>m</i>). • <i>x.n.ATTRIBUTEVALUE.m.CPUID</i> is the <i>m</i>-th CPUID value. • <i>x.n.ATTRIBUTEVALUE.m.PSW</i> is the <i>m</i>-th PSW value. <p>Note: The input connection token must represent an <i>image connection</i>.</p>	Field Name	Field Type	-----	-----	Number of CPs	32-bit unsigned integer
Field Name	Field Type						
-----	-----						
Number of CPs	32-bit unsigned integer						
92 (146) HWI_GROUP_PROFILE_CAPACITY	<p>Requests to change or set the workload unit capacity for the group profile associated with an image.</p> <p>Note:</p> <ol style="list-style-type: none"> 1. The input connection token must only represent an <i>image connection</i>. 2. This attribute requires that the target image be: <ul style="list-style-type: none"> • On a z196 (zEnterprise) or higher CPC. • A member of a LPAR (defined capacity) group. <p>If both of these requirements are not met, the HWIQUERY fails with RC=X'406' (HWI_QUERY_ATTRIBUTE_NOT_SUPPORTED).</p>						
93 (147) HWI_LAST_USED_LOADADDR	<p>Requests to retrieve the load address that was used when the image was last loaded. This attribute is only available when targeting an image residing on a z196 or later CPC.</p> <p>Note: The input connection token must represent an <i>image connection</i>.</p>						
94 (148) HWI_LAST_USED_LOADPARAM	<p>Requests to retrieve the load parameters that were used when the image was last loaded. This attribute is only available when targeting an image residing on a z196 or later CPC.</p> <p>Note: The input connection token must represent an <i>image connection</i>.</p>						
95-B6 (149-182) RESERVED	<p>Additional attributes and reserved numbers for attributes that are for <i>image connections</i> only.</p>						
B7 (183) HWI_RECID	<p>Requests to retrieve the record ID.</p> <p>Note: The input connection token must only represent a <i>capacity record connection</i>.</p>						

HWIQUERY

Constant in hexadecimal (Decimal) Equate symbol	Description
B8 (184) HWI_RECTYPE	Requests to retrieve the record type. Note: The input connection token must only represent a <i>capacity record connection</i> .
B9 (185) HWI_ACTSTAT	Requests to retrieve the record activation status. Note: The input connection token must only represent a <i>capacity record connection</i> .
BA (186) HWI_ACTDATE	Requests to retrieve the record activation date. Note: The input connection token must only represent a <i>capacity record connection</i> .
BB (187) HWI_EXPDATE	Requests to retrieve the record expiration date. Note: The input connection token must only represent a <i>capacity record connection</i> .
BC (188) HWI_ACTEXP	Requests to retrieve the record activation expiration date. Note: The input connection token must only represent a <i>capacity record connection</i> .
BD (189) HWI_MAXRADS	Requests to retrieve the maximum real activation days. Note: The input connection token must only represent a <i>capacity record connection</i> .
BE (190) HWI_MAXTADS	Requests to retrieve the maximum test activation days. Note: The input connection token must only represent a <i>capacity record connection</i> .
BF (191) HWI_REMRADS	Requests to retrieve the remaining real activation days. Note: The input connection token must only represent a <i>capacity record connection</i> .
C0 (192) HWI_REMTADS	Requests to retrieve the remaining test activation days. Note: The input connection token must only represent a <i>capacity record connection</i> .
C1 (193) HWI_OOCODREC	Request to retrieve all aspects of a capacity record in XML format. Note: The input connection token must only represent a <i>capacity record connection</i> .
C3-C8 (195-200) RESERVED	Reserved for capacity record attributes.
C9 (201) HWI_IOCDS	Requests to retrieve the IOCDS. Note: The input connection token must represent a <i>reset activation profile</i> .

Constant in hexadecimal (Decimal) Equate symbol	Description
CA (202) HWI_IPL_ADDRESS	Requests to retrieve the IPL address. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CB (203) HWI_IPL_PARM	Requests to retrieve the IPL parameter. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CC (204) HWI_IPL_TYPE	Requests to retrieve the IPL type for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CD (205) HWI_WW_PORTNAME	Requests to retrieve the worldwide port name for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CE (206) HWI_BOOT_PGM_SELECTOR	Requests to retrieve the boot program selector for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CF (207) HWI_LU_NUM	Requests to retrieve the logical unit number value for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
D0 (208) HWI_BOOTREC_BLK_ADDR	Requests to retrieve the boot record logical block address for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
D1 (209) HWI_OPSYS_LOADPARM	Requests to retrieve the operating system specific load parameter. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
D2 (210) HWI_GROUP_PROF_NAME	Requests to retrieve the name of the group capacity profile that is to be used for the CPC image or image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
D3 (211) HWI_LOAD_AT_ACTIVATION	Requests to retrieve the indicator if the CPC image object activated with this profile should be loaded (IPLed) at the end of the activation. Note: The input connection token must represent an <i>image activation profile</i> .
D4 (212) HWI_CENTRAL_STOR	Requests to retrieve the initial amount of central storage (in megabytes) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
D5 (213) HWI_RES_CENTRAL_STOR	Requests to retrieve the reserved amount of central storage (in megabytes) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .

HWIQUERY

Constant in hexadecimal (Decimal) Equate symbol	Description
D6 (214) HWI_EXPANDED_STOR	Requests to retrieve the initial amount of expanded storage (in megabytes) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
D7 (215) HWI_RES_EXPANDED_STOR	Requests to retrieve the reserved amount of expanded storage (in megabytes) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
D8 (216) HWI_NUM_GPP	Requests to retrieve the number of dedicated general purpose processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
D9 (217) HWI_NUM_RESGPP	Requests to retrieve the number of reserved dedicated general purpose processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DA (218) HWI_NUM_IFA	Requests to retrieve the number of dedicated integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DB (219) HWI_NUM_RESIFA	Requests to retrieve the number of reserved dedicated integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DC (220) HWI_NUM_IFL	Requests to retrieve the number of dedicated integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DD (221) HWI_NUM_RESIFL	Requests to retrieve the number of reserved dedicated integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DE (222) HWI_NUM_ICF	Requests to retrieve the number of dedicated internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DF (223) HWI_NUM_RESICF	Requests to retrieve the number of reserved dedicated internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E0 (224) HWI_NUM_ZIIP	Requests to retrieve the number of dedicated System z Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E1 (225) HWI_NUM_RESZIIP	Requests to retrieve the number of reserved dedicated System z Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .

Constant in hexadecimal (Decimal) Equate symbol	Description
E2 (226) HWI_NUM_SHARED_GPP	Requests to retrieve the number of shared general purpose processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E3 (227) HWI_NUM_RES_SHARED_GPP	Requests to retrieve the number of reserved shared general purpose processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E4 (228) HWI_NUM_SHARED_IFA	Requests to retrieve the number of shared integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E5 (229) HWI_NUM_RES_SHARED_IFA	Requests to retrieve the number of reserved shared integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E6 (230) HWI_NUM_SHARED_IFL	Requests to retrieve the number of shared integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E7 (231) HWI_NUM_RES_SHARED_IFL	Requests to retrieve the number of reserved shared integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E8 (232) HWI_NUM_SHARED_ICF	Requests to retrieve the number of shared internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E9 (233) HWI_NUM_RES_SHARED_ICF	Requests to retrieve the number of reserved shared internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
EA (234) HWI_NUM_SHARED_ZIIP	Requests to retrieve the number of shared System z Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
EB (235) HWI_NUM_RES_SHARED_ZIIP	Requests to retrieve the number of reserved shared System z Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
EC (236) HWI_BASIC_CPU_AUTH _COUNT_CNTL	Requests to retrieve the enablement value of the Basic CPU counter facility for the CPC image. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .

HWIQUERY

Constant in hexadecimal (Decimal) Equate symbol	Description
ED (237) HWI_PROBSTATE_CPU_AUTH _COUNT_CNTL	Requests to retrieve the enablement value of the Problem state CPU counter facility for the CPC image. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
EE (238) HWI_CRYPTACTIVITY_CPU _AUTH_COUNT_CNTL	Requests to retrieve the enablement value of the crypto activity CPU counter facility for the CPC image. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
EF (239) HWI_EXTENDED_CPU_AUTH _COUNT_CNTL	Requests to retrieve the enablement value of the extended CPU counter facility for the CPC image. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F0 (240) HWI_COPROCESSOR_CPU _AUTH_COUNT_CNTL	Requests to retrieve the enablement value of the coprocessor group CPU counter facility for the CPC image. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F1 (241) HWI_BASIC_CPU_SAMPLING _AUTH_CNTL	Requests to retrieve the enablement value of the basic CP CPU sampling facility for the CPC image. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F2 (242) HWI_APROF_STORE_STATUS	Requests to retrieve the store status function value. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent a <i>load activation profile</i> .
F3 (243) HWI_APROF_LOADTYPE	Requests to retrieve the type of load being requested. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent a <i>load activation profile</i> .
F4 (244) HWI_PROFILE_DESCRIPTION	Requests to retrieve the activation profile description. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F5 (245) HWI_PROFILE_PARTITION _ID	Requests to retrieve the partition identifier for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F6 (246) HWI_OPERATING_MODE	Requests to retrieve the operating mode value for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .

Constant in hexadecimal (Decimal) Equate symbol	Description
F7 (247) HWI_CLOCK_TYPE	Requests to retrieve the clock type assignment (time source setting) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F8 (248) HWI_TIME_OFFSET_DAYS	Requests to retrieve the time offset days (the number of days currently set as the offset from the external time source's time of day) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F9 (249) HWI_TIME_OFFSET_HOURS	Requests to retrieve the time offset hours (the number of hours currently set as the offset from the external time source's time of day) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FA (250) HWI_TIME_OFFSET _MINUTES	Requests to retrieve the time offset minutes (the number of minutes currently set as the offset from the external time source's time of day) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FB (251) HWI_TIME_OFFSET _INCREASE	Requests to retrieve the time offset increase or decrease value for the activation profile. The time offset, as specified in days, hours, and minutes, is increased or decreased from GMT. TRUE means that the time offset is east of GMT. FALSE means that the time offset is west of GMT. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FC (252) HWI_LICCC_VALIDATION _ENABLED	Requests to retrieve whether the activation profile must conform to the current Licensed Internal Code Configuration Control (LICCC) configuration. This attribute is only available when targeting a zEnterprise or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FD (253) HWI_GLOBAL _PERFORMANCE _DATA_CONTROL	Requests to retrieve whether the logical partition can be used to view the processing unit activity data for all other LPARs activated on the same CPC. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FE (254) HWI_IO_CONFIGURATION _CONTROL	Requests to retrieve whether the logical partition can be used to read and write any Input/Output Configuration Data Set (IOCDs) in the configuration. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FF (255) HWI_CROSS_PARTITION _AUTHORITY	Requests to retrieve whether the logical partition can be used to issue control program instructions that reset or deactivate other LPARs. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .

HWIQUERY

Constant in hexadecimal (Decimal) Equate symbol	Description
100 (256) HWI_LOGICAL_PARTITION _ISOLATION	Requests to retrieve whether reconfigurable channel paths assigned to the logical partition are reserved for its exclusive use. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
101-109 (257-265) RESERVED	Reserved for activation profile attributes.

NumOfAttributes (non-REXX)

Supplied parameter

- Type: Integer
- Length: 4 bytes

NumOfAttributes specifies the number of attributes to be queried. The maximum number of attributes allowed is 64.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the DiagArea might not be filled in, and the data returned in the area should be ignored.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code that is returned from the console application API or the BCPii transport layer.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, “BCPii communication error reason codes,” on page 583 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0006yyyy' because of one of the following reasons:

Table 66. Reasons for abend X'042', RC X'0006yyyy'

yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See *z/OS MVS System Codes* for additional information.

Return codes

When the service returns control to the caller, GPR 15 and ReturnCode contain a hexadecimal return code.

Return Code in Hexadecimal Equate Symbol	Meaning and Action
0 HWI_OK	<p>Meaning: Successful completion.</p> <p>Action: None.</p>
100 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified connect token is not valid. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The connect token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken. • The connect token does not represent an active connection. The connection specified might have already been disconnected using the HWIDISC service call. • The connect token is not associated with the address space of the caller. The ConnectToken specified is associated with a different address space than the caller of this service call. <p>Action: Check for probable coding error.</p>
101 HWI_COMMUNICATION_ERROR	<p>Meaning: A communication error is detected. The hardware management console application API (HWMCA) or the BCPii transport layer has returned with a failing return code.</p> <p>Action: See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPii transport layer. In some cases, the Diag_Index and Diag_Key may contain additional details.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, "BCPii communication error reason codes," on page 583.</p>

HWIQUERY

Return Code in Hexadecimal Equate Symbol	Meaning and Action
102 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify that the specified DiagArea is defined as a 32-byte character field.</p>
103 HWI_CONNECT_TOKEN_INACTIVE	<p>Meaning: The specified connect token is no longer valid. The connection has been disconnected or it is in the progress of being disconnected.</p> <p>Action: Check for probable coding error. Verify that the specified connect token is still active. If connectivity to the targeted CPC connection no longer exists, all connections associated with that CPC will no longer have a connect token that can be used.</p>
401 HWI_QUERYPARM_ATTRIB_INV	<p>Meaning: Program error. One of the requested attribute identifiers in the QueryParm is not valid. The system rejects the service call. This return code indicates that one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The Query attribute identifier specified is not in the acceptable value range of possible attributes. • The specified Query attribute identifier has been provided with an incompatible connection type. For example, the attribute identifier applies only to CPC connections, but the ConnectToken specified represents an image connection, a capacity record connection, or any of the activation profile connections. <p>Action: Check for probable coding error. Validate that the Query attribute specified is in the valid range of possible values. Validate that the Query attribute specified is permitted for the specified connection type.</p> <p>See the DiagArea for further diagnostic information:</p> <ul style="list-style-type: none"> • The Diag_Index field specifies the index of the element in the attribute array that is in error. • The Diag_Key contains the attribute identifier specified. • The Diag_Text contains "Invalid Attr" if the attribute is one whose value cannot be queried. If the attribute cannot be queried for the specified connection type, the Diag_Text contains "Mismatch."
402 HWI_QUERYPARM_INACCESSIBLE	<p>Meaning: Program error. The QueryParm data area is either partially or completely inaccessible by the application, the Base Control Program internal interface (BCPii) address space, or both.</p> <p>Action: Check for probable coding error. Consider the following possibilities:</p> <ul style="list-style-type: none"> • The QueryParm length could be too small. The size of QueryParm must be at least the product of the NumofAttributes parameter and the length of the data area mapping for each attribute (16 bytes). • The NumofAttributes value can be larger than the number of parameters actually passed.

Return Code in Hexadecimal Equate Symbol	Meaning and Action
403 HWI_QUERYPARM_ATTRIBRETADDR _INACCESSIBLE	<p>Meaning: Program error. Storage that is pointed to by one or more of the attribute value pointers in the QueryParm is not accessible by the application. The system is not able to return data for this attribute identifier. Partial data might have already been returned.</p> <p>Action: Check for probable coding error. See the DiagArea for further diagnostic information. The Diag_Index field specifies the array index that contained the inaccessible AttributeValuePtr. The Diag_Key contains the erroneous attribute identifier.</p>
404 HWI_QUERYPARM_ATTRIB_LENGTH_ INV	<p>Meaning: Program error. One of the attribute lengths specified is too small. There is not enough space to contain all of the returned data for this particular attribute. The system returns partial data, filling in the attribute data area for the length specified.</p> <p>Action: Check for probable coding error. See the DiagArea for further diagnostic information. The Diag_Index field specifies the array index which contained the partially filled-in value. The Diag_Key is the attribute identifier constant that causes the error. The Diag_Actual indicates the application-specified length. The Diag_Expected indicates the size required for the returned data.</p>
405 HWI_QUERY_NUMOFATTRIB_INV	<p>Meaning: Program error. The NumOfAttributes specified on the call is not valid. The NumOfAttributes value must be in the range of 1 to 64.</p> <p>Action: Check for probable error. Verify that the NumOfAttributes specified is greater than zero and less than or equal to 64.</p>
406 HWI_QUERY_ATTRIBUTE_NOT SUPPORTED	<p>Meaning: The targeted hardware of the HWIQUERY request does not recognize the attribute attempted to be retrieved.</p> <p>Action: Verify that the targeted hardware is at a level that supports the type of attribute being queried.</p>
407 HWI_QUERY_TARGET_DEACTIVATED	<p>Meaning: A query attribute could not be retrieved because the targeted object is deactivated.</p> <p>Action: Verify that the targeted object is activated. Activate the object before attempting to retrieve this same attribute again.</p>
408 HWI_QUERY_ATTRIB_TEMP_NOT_AVAILABLE	<p>Meaning: One or more query attributes could not be retrieved because the support element (SE) is temporarily unavailable.</p> <p>Action: Try this request again at a later time. If the problem persists, contact the IBM Support Center.</p>

HWIQUERY

Return Code in Hexadecimal Equate Symbol	Meaning and Action
F00 HWI_NOT_AVAILABLE	<p>Meaning: BCPii services are not available, and the system rejects the service request.</p> <p>Action: Notify the system programmer to start the BCPii address space and try the request again. See "Restarting the HWIBCPii address space" on page 269 about how to start the BCPii address space.</p> <p>Programs can also listen to ENF68 to determine when BCPii services are available. See <i>z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG</i> for how to listen for BCPii activation messages.</p>
F01 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state and the program does not reside in an APF-authorized library.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for CPC, activation profile, or user-defined image group connections. • Define read access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for an image connections. • Define read access authorization to the FACILITY class resource profile HWI.CAPREC.netid.nau.caprecid for a capacity record connection. • Ensure that the referenced facility class profile is RACLIST-specified.
F03 HWI_INTERRUPT_STATUS_INV	<p>Meaning: The calling program is disabled. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWI_MODE_INV	<p>Meaning: The calling program is not in task mode. The system rejects this service request.</p> <p>Action: Check the calling program for a probable error.</p>
F05 HWI_LOCKS_HELD	<p>Meaning: The calling program is holding one or more locks. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F06 HWI_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects this service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports BCPii services. Then run the calling program again.</p>

Return Code in Hexadecimal Equate Symbol	Meaning and Action
F07 HWI_UNSUPPORTED_ENVIRONMENT	<p>Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine).</p> <p>Action: Issue the BCPii service from a different execution environment.</p>
FFF HWI_UNEXPECTED_ERROR	<p>Meaning: System error. The service that was called encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Example

In the pseudocode example, the caller issues a call to retrieve the CPC name and the Current CPC status of a CPC:

```

.
QueryParm_Ptr = ADDR(QueryParm);
NumberOfAttributes = 2;
QueryParm(1).AttributeIdentifier = HWI_NAME;
QueryParm(1).AttributeValue_Ptr = Addr(Value1);
QueryParm(1).AttributeValueLen = length of value1;
QueryParm(2).AttributeIdentifier = HWI_OPERSTAT;
QueryParm(2).AttributeValue_Ptr = Addr(Value2);
QueryParm(2).AttributeValueLen = 4;
CALL HWIQUERY (ReturnCode, ConnectToken, QueryParm_Ptr,
              NumOfAttributes, DiagArea)
.

```

A REXX programming example for the HWIQUERY service:

```

myQueryParm.0 = 4 /* Set number of attributes */
myQueryParm.n.ATTRIBUTEIDENTIFIER = HWI_NAME
myQueryParm.n.ATTRIBUTEIDENTIFIER = HWI_LUAPROF
myQueryParm.n.ATTRIBUTEIDENTIFIER = HWI_MSERIAL
myQueryParm.n.ATTRIBUTEIDENTIFIER = HWI_IPADDR

address bcpii "hwiquery RetCode myConnectToken myQueryParm. myDiag."

If (RC <> 0) | (Retcode <> 0) Then
  Do
    Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
    If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
      Do
        Say ' Diag_index=' myDiag.DIAG_INDEX
        Say ' Diag_key=' myDiag.DIAG_KEY
        Say ' Diag_actual=' myDiag.DIAG_ACTUAL
        Say ' Diag_expected=' myDiag.DIAG_EXPECTED
        Say ' Diag_commer=' myDiag.DIAG_COMMERR
        Say ' Diag_text=' myDiag.DIAG_TEXT
      End
    End
Else
  Do n=1 to myQueryParm.0
    Say ' myQueryParm.'n'.ATTRIBUTEVALUE = 'myQueryParm.n.ATTRIBUTEVALUE
  End

```

HWISET — BCPii set SE/HMC-managed attributes

Call the HWISET service to change or set data for Hardware Management Console (HMC)-managed objects associated with Central Processor Complexes (CPCs), CPC images (LPARs), or activation profiles.

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	PKM allowing key 0-7, or supervisor state
Dispatchable unit mode:	Task
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:	Control parameters must be in the primary address space and addressable by the caller
Linkage:	Standard MVS linkage conventions are used

Programming requirements

See “Syntax, linkage and programming considerations” on page 271 for details about how to call BCPii services in the various programming languages.

See “HWIQUERY and HWISET” on page 585 for the summary table of the BCPii HWIQUERY and HWISET attributes and the objects that can be targeted for each function.

REXX programming considerations for the HWISET service

All information for the HWISET service applies for REXX requests except:

- SetTypeValue replaces SetTypeValue_Ptr. The actual value to be set, represented in character form, is passed instead of a pointer.
- The SetTypeValueLen input parm is not used.

Restrictions

BCPii does not allow HWISET to be issued from within a BCPii ENF exit routine.

Authorization

The client application must have at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV. This class resource grants the application access to consult to the local CPC.

In addition, the client application must have at least update access to the SAF-protected FACILITY class resource profile HWI.TARGET.*netid.nau* for setting CPC-related or activation profile-related values, or HWI.TARGET.*netid.nau.imagename* for setting image-related values.

Note: BCPii requires the FACILITY class to be RACLIST-specified.

SMF recording

Requests that complete with a return code of zero will have SMF type 106 (X'6A') records written if the installation has activated recording of this record type in its active configuration.

Syntax

Write the call as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Non-REXX parameters	REXX parameters
CALL HWISET(ReturnCode, ConnectToken, SetType, SetTypeValue_Ptr SetTypeValueLen, DiagArea);	address bcpII "hwiset ReturnCode ConnectToken SetType SetTypeValue DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

ReturnCode contains the return code from the service.

ConnectToken

Supplied parameter

- Type: Character string
- Length: 16 bytes

ConnectToken represents a logical connection between the application and a CPC, image, or activation profile. **ConnectToken** is an output parameter on the HWICONN service call.

In most cases, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call. For BCPii REXX execs running under the TSO/E or ISV-provided REXX environments, the ConnectToken specified must have originated from a HWICONN service call that was issued from the same task.

SetType

Supplied parameter

- Type: Integer (non-REXX), character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

SetType specifies the type of set request.

The following table is the list of valid set types. See the following documentation for more information:

- *System z Application Programming Interfaces* (SB10-7030-13)
- *System z10 and eServer zSeries Application Programming Interfaces* (SB10-7030-09)
- *System z9 and eServer zSeries Application Programming Interfaces* (SB10-7030-08)

HWISET

- Publication appropriate to the level of hardware that the HWISET is targeted.

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
6 (6) HWI_ACCSTAT	Requests to change or set the acceptable CPC status values. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
7 (7) HWI_APROF	Requests to change or set the next activation reset profile name. Note: The input connection token represents a <i>CPC connection</i> or an <i>image connection</i> .
27 (39) HWI_PRUNTYPE	Requests to change or set the processor running time type. Note: The input connection token represents a <i>CPC connection</i> or a <i>reset activation profile connection</i> .
28 (40) HWI_PRUNTIME	Requests to change or set the processor running time type. Note: The input connection token must only represent a <i>CPC connection</i> or a <i>reset activation profile connection</i> .
29 (41) HWI_PRUNTSEW	Requests to change or set the processor running time slice end wait processing. Note: The input connection token must only represent a <i>CPC connection</i> or a <i>reset activation profile connection</i> .
70 (112) HWI_DEFCAP	Requests to change or set the current defined capacity. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
71 (113) HWI_SGPIPW	Requests to change or set the shared general processor initial processing weight (SGPIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
72 (114) HWI_SGPIPWCAP	Requests to change or set the SGPIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
73 (115) HWI_SGPPWMIN	Requests to change or set the minimum SGPPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
74 (116) HWI_SGPPWMAX	Requests to change or set the maximum SGPPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
77 (119) HWI_WLM	Requests to change or set whether WLM is allowed to change SGPPW values. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
78 (120) HWI_IFAIPW	Requests to change or set the integrated facility for applications initial processing weight (IFAIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
79 (121) HWI_IFAIPWCAP	Requests to change or set the IFAIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7A (122) HWI_IFAPWMIN	Requests to change or set the minimum IFAPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7B (123) HWI_IFAPWMAX	Requests to change or set the maximum IFAPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7E (126) HWI_IFLIPW	Requests to change or set the integrated facility for Linux initial processing weight (IFLIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
7F (127) HWI_IFLIPWCAP	Requests to change or set the IFLIPW to be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
80 (128) HWI_IFLPWMIN	Requests to change or set the minimum IFLPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
81 (129) HWI_IFLPWMAX	Requests to change or set the maximum IFLPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
84 (132) HWI_ICFIPW	Requests to change or set the internal coupling facility initial processing weight (ICFIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
85 (133) HWI_ICFIPWCAP	Requests to change or set the ICFIPW be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
86 (134) HWI_ICFPWMIN	Requests to change or set the minimum ICFPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .

HWISET

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
87 (135) HWI_ICFPWMAX	Requests to change or set the maximum ICFPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8A (138) HWI_IIPW	Requests to change or set the integrated information processors initial processing weight (IIPW). Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8B (139) HWI_IIPWCAP	Requests to change or set the IIPW be capped or not capped. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8C (140) HWI_IIPWMIN	Requests to change or set the minimum IIPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
8D (141) HWI_IIPWMAX	Requests to change or set the maximum IIPW value. Note: The input connection token must only represent an <i>image connection</i> or an <i>image activation profile connection</i> .
92 (146) HWI_GROUP_PROFILE_CAPACITY	Requests to change or set the workload unit capacity for the group profile associated with an image. Note: <ol style="list-style-type: none"> The input connection token must only represent an <i>image connection</i>. This attribute requires the target image be: <ul style="list-style-type: none"> On a z196 (zEnterprise) or higher CPC. A member of a LPAR (defined capacity) group. <p>If both of these requirements are not met, the HWISET fails with RC=X'101' (HWI_COMMUNICATON_ERROR), with the DiagCommErr value set to X'15' (21) (HWMCA_DE_SNMP_ERROR).</p>
C9 (201) HWI_IOCDS	Requests to change or set the IOCDS. Note: The input connection token must represent a <i>reset activation profile</i> .
CA (202) HWI_IPL_ADDRESS	Requests to change or set the IPL address. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CB (203) HWI_IPL_PARM	Requests to change or set the IPL parameter. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
CC (204) HWI_IPL_TYPE	Requests to change or set the IPL type for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CD (205) HWI_WW_PORTNAME	Requests to change or set the worldwide port name for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CE (206) HWI_BOOT_PGM_SELECTOR	Requests to change or set the boot program selector for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
CF (207) HWI_LU_NUM	Requests to change or set the logical unit number value for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
D0 (208) HWI_BOOTREC_BLK_ADDR	Requests to change or set the boot record logical block address for the activation profile. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
D1 (209) HWI_OPSYS_LOADPARAM	Requests to change or set the operating system specific load parameter. Note: The input connection token must represent an <i>image activation profile</i> or a <i>load activation profile</i> .
D2 (210) HWI_GROUP_PROF_NAME	Requests to change or set the name of the group capacity profile that is to be used for the CPC image or image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
D3 (211) HWI_LOAD_AT_ACTIVATION	Requests to change or set the indicator if the CPC image object activated with this profile should be loaded (IPLed) at the end of the activation. Note: The input connection token must represent an <i>image activation profile</i> .
D4 (212) HWI_CENTRAL_STOR	Requests to change or set the initial amount of central storage (in megabytes) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
D5 (213) HWI_RES_CENTRAL_STOR	Requests to change or set the reserved amount of central storage (in megabytes) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
D6 (214) HWI_EXPANDED_STOR	Requests to change or set the initial amount of expanded storage (in megabytes) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .

HWISET

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
D7 (215) HWI_RES_EXPANDED_STOR	Requests to change or set the reserved amount of expanded storage (in megabytes) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
D8 (216) HWI_NUM_GPP	Requests to change or set the number of dedicated general purpose processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
D9 (217) HWI_NUM_RESGPP	Requests to change or set the number of reserved dedicated general purpose processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DA (218) HWI_NUM_IFA	Requests to change or set the number of dedicated integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DB (219) HWI_NUM_RESIFA	Requests to change or set the number of reserved dedicated integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DC (220) HWI_NUM_IFL	Requests to change or set the number of dedicated integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DD (221) HWI_NUM_RESIFL	Requests to change or set the number of reserved dedicated integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DE (222) HWI_NUM_ICF	Requests to change or set the number of dedicated internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
DF (223) HWI_NUM_RESICF	Requests to change or set the number of reserved dedicated internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E0 (224) HWI_NUM_ZIIP	Requests to change or set the number of dedicated System z Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
E1 (225) HWI_NUM_RESZIIP	Requests to change or set the number of reserved dedicated System z Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E2 (226) HWI_NUM_SHARED_GPP	Requests to change or set the number of shared general purpose processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E3 (227) HWI_NUM_RES_SHARED_GPP	Requests to change or set the number of reserved shared general purpose processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E4 (228) HWI_NUM_SHARED_IFA	Requests to change or set the number of shared integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E5 (229) HWI_NUM_RES_SHARED_IFA	Requests to change or set the number of reserved shared integrated facility for applications (IFA) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E6 (230) HWI_NUM_SHARED_IFL	Requests to change or set the number of shared integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E7 (231) HWI_NUM_RES_SHARED_IFL	Requests to change or set the number of reserved shared integrated facility for Linux (IFL) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E8 (232) HWI_NUM_SHARED_ICF	Requests to change or set the number of shared internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
E9 (233) HWI_NUM_RES_SHARED_ICF	Requests to change or set the number of reserved shared internal coupling facility (ICF) processors to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
EA (234) HWI_NUM_SHARED_ZIIP	Requests to change or set the number of shared System z Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .

HWISET

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
EB (235) HWI_NUM_RES_SHARED _ZIIP	Requests to change or set the number of reserved shared System z Integrated Information Processors (zIIPs) to be used for the CPC image object activated with this profile. Note: The input connection token must represent an <i>image activation profile</i> .
EC (236) HWI_BASIC_CPU_AUTH _COUNT_CNTL	Requests to change or set the enablement value of the basic CPU counter facility for the CPC image object activated with this profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
ED (237) HWI_PROBSTATE_CPU _AUTH_COUNT_CNTL	Requests to change or set the enablement value of the Problem state CPU counter facility for the CPC image object activated with this profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
EE (238) HWI_CRYPTOACTIONIVITY_CPU _AUTH_COUNT_CNTL	Requests to change of set the enablement value of the crypto activity CPU counter facility for the CPC image object activated with this profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
EF (239) HWI_EXTENDED_CPU _AUTH_COUNT_CNTL	Requests to change or set the enablement value of the extended CPU counter facility for the CPC image object activated with this profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F0 (240) HWI_COPROCESSOR_CPU _AUTH_COUNT_CNTL	Requests to change or set the enablement value of the coprocessor group CPU counter facility for the CPC image object activated with this profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F1 (241) HWI_BASIC_CPU_SAMPLING _AUTH_CNTL	Requests to change or set the enablement value of the basic CP CPU sampling facility for the CPC image object activated with this profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F2 (242) HWI_APROF_STORE_STATUS	Requests to change or set the store status function value. This value is only valid if HWI_APROF_LOADTYPE is set to normal. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent a <i>load activation profile</i> .

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
F3 (243) HWI_APROF_LOADTYPE	Requests to change or set the type of load being requested. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent a <i>load activation profile</i> .
F4 (244) HWI_PROFILE_DESCRIPTION	Requests to change or set the activation profile description. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F5 (245) HWI_PROFILE_PARTITION_ID	Requests to change or set the partition identifier for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F6 (246) HWI_OPERATING_MODE	Requests to change or set the operating mode value for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F7 (247) HWI_CLOCK_TYPE	Requests to change or set the clock type assignment (time source setting) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F8 (248) HWI_TIME_OFFSET_DAYS	Requests to change or set the time offset days (the number of days currently set as the offset from the external time source's time of day) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
F9 (249) HWI_TIME_OFFSET_HOURS	Requests to change or set the time offset hours (the number of hours currently set as the offset from the external time source's time of day) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FA (250) HWI_TIME_OFFSET_MINUTES	Requests to change or set the time offset minutes (the number of minutes currently set as the offset from the external time source's time of day) for the activation profile. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FB (251) HWI_TIME_OFFSET_INCREASE	Requests to change or set the time offset increase or decrease value for the activation profile. The time offset, as specified in days, hours, and minutes, is increased or decreased from GMT. TRUE means that the time offset is east of GMT. FALSE means that the time offset is west of GMT. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .

HWISET

Constant in: Hexadecimal, (Decimal), Equate symbol	Description
FC (252) HWI_LICCC_VALIDATION _ENABLED	Requests to change or set whether the activation profile must conform to the current Licensed Internal Code Configuration Control (LICCC) configuration. This attribute is only available when targeting a zEnterprise or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FD (253) HWI_GLOBAL_PERFORMANCE _DATA_CONTROL	Requests to change or set whether the logical partition can be used to view the processing unit activity data for all other LPARs activated on the same CPC. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
FE (254) HWI_IO_CONFIGURATION _CONTROL	Requests to change or set whether the logical partition can be used to read and write any Input/Output Configuration Data Set (IOCDs) in the configuration. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
100 (256) HWI_LOGICAL_PARTITION _ISOLATION	Requests to change or set whether reconfigurable channel paths assigned to the logical partition are reserved for its exclusive use. This attribute is only available when targeting a z10 or higher CPC. Note: The input connection token must represent an <i>image activation profile</i> .
101-109 (257-264) RESERVED	Reserved for activation profile attributes.

SetTypeValue_Ptr (non-REXX)

SetTypeValue (REXX)

Supplied parameter

- Type: Pointer (non-REXX), character or character representation of an integer (REXX)
- Length: 4 bytes (non-REXX)

Non-REXX:

SetTypeValue_Ptr specifies the address of the value to be set or changed. Some **SetType** requests allow a null value to be set. If a null value is desired, the **SetTypeValue_Ptr** value must be zero.

REXX:

SetTypeValue specifies the value to be set or changed. Some **SetType** requests allow a null value to be set. If a null value is desired, **SetTypeValue** should be set to null ("").

The particular **SetType** determines what data value must be specified. See the following chart as well as the following documentation for more information:

- *System z Application Programming Interfaces (SB10-7030-13)*
- *System z10 and eServer zSeries Application Programming Interfaces (SB10-7030-09)*

- System z9 and eServer zSeries Application Programming Interfaces (SB10-7030-08)

SetType values in hexadecimal, (decml), and equate symbol	Values to be specified
<p>6 (6) HWI_ACCSTAT</p>	<p>A 4-byte integer type value.</p> <p>For CPC connections, bit values can be set to:</p> <ul style="list-style-type: none"> • HWMCA_STATUS_OPERATING • HWMCA_STATUS_NOT_OPERATING • HWMCA_STATUS_NO_POWER • HWMCA_STATUS_EXCEPTIONS • HWMCA_STATUS_STATUS_CHECK • HWMCA_STATUS_SERVICE • HWMCA_STATUS_LINKNOTACTIVE • HWMCA_STATUS_POWERSAVE • HWMCA_STATUS_SERVICE_REQ • HWMCA_STATUS_DEGRADED <p>For image connections, bit values can be set to:</p> <ul style="list-style-type: none"> • HWMCA_STATUS_OPERATING • HWMCA_STATUS_NOT_OPERATING • HWMCA_STATUS_NOT_ACTIVATED • HWMCA_STATUS_EXCEPTIONS • HWMCA_STATUS_STATUS_CHECK • HWMCA_STATUS POWERSAVE
<p>7 (7) HWI_APROF</p>	<p>A 16-character activation profile name padded with trailing blanks.</p>
<p>27 (39) HWI_PRUNTYPE</p>	<p>A 4-byte integer type value.</p> <p>HWMCA_DETERMINED_SYSTEM The processor running is dynamically determined by the system.</p> <p>HWMCA_DETERMINED_USER The processor running time is set to a constant value.</p>
<p>28 (40) HWI_PRUNTIME</p>	<p>A 4-byte integer type value.</p> <p>A value between 1 to 100 for the user defined processor running time.</p> <p>Note: This value can only be set if the processor running time type (HWI_PRUNTYPE) is set to HWMCA_DETERMINED_USER.</p>

HWISET

SetType values in hexadecimal, (decml), and equate symbol	Values to be specified
29 (41) HWI_PRUNTSEW	A 4-byte integer type value. HWMCA_TRUE Indicates that an image should lose its share of running time when it enters a wait state. HWMCA_FALSE Indicates that an image should not lose its share of running time when it enters a wait state. Note: This value can only be set if the processor running time type (HWI_PRUNTYPE) is set to HWMCA_DETERMINED_USER.
70 (112) HWI_DEFCAP	A 4-byte integer type value. A value represents the amount of defined capacity specified for the logical partition. A value of 0 indicates that no defined capacity is specified for the logical partition.
71 (113) HWI_SGPIPW	A 4-byte integer type value. A value from 1 - 999 defines the relative amount of shared general purpose processor resources allocated to the CPC image object. A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated general purpose processor. Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated general purpose processor.
72 (114) HWI_SGPIPWCAP	A 4-byte integer type value. This indicates that the initial general purpose processor processing weight for the CPC image object is capped or not capped. HWMCA_TRUE Capped HWMCA_FALSE Not capped
73 (115) HWI_SGPPWMIN	A 4-byte integer type value. A value from 1 - 999 and less than or equal to the initial processing weight defines the minimum relative amount of shared general purpose processor resources allocated to the CPC image object. A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated general purpose processor. Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated general purpose processor.

SetType values in hexadecimal, (decml), and equate symbol	Values to be specified
<p>74 (116) HWI_SGPPWMAX</p>	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 and greater than or equal to the initial processing weight defines the maximum relative amount of shared general purpose processor resources allocated to the CPC image object.</p> <p>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated general purpose processor. Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated general purpose processor.</p>
<p>77 (119) HWI_WLM</p>	<p>A 4-byte integer type value.</p> <p>This indicates whether the Workload Manager is allowed to change processor weight-related attributes.</p> <ul style="list-style-type: none"> • HWMCA_TRUE • HWMCA_FALSE <p>HWI_WLM must be set to HWMCA_TRUE before any of the settings for the specialized IFA, IFL, ICF, or IIP engines can be modified.</p>
<p>78 (120) HWI_IFAIPW</p>	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the relative amount of shared integrated facility for applications (IFA) processor resources allocated to the CPC image object.</p> <p>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor. Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor.</p>
<p>79 (121) HWI_IFAIPWCAP</p>	<p>A 4-byte integer type value. This indicates whether the initial processing weight for integrated facility for applications (IFA) processors is a limit or a target.</p> <p>HWMCA_TRUE Capped</p> <p>HWMCA_FALSE Not capped</p>

HWISET

SetType values in hexadecimal, (decial), and equate symbol	Values to be specified
7A (122) HWI_IFAPWMIN	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the minimum relative amount of shared integrated facility for applications (IFA) processor resources allocated to the CPC image object.</p> <p>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor. Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor.</p>
7B (123) HWI_IFAPWMAX	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the maximum relative amount of shared integrated facility for applications (IFA) processor resources allocated to the CPC image object.</p> <p>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor. Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated facility for applications (IFA) processor.</p>
7E (126) HWI_IFLIPW	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the relative amount of shared integrated facility for Linux (IFL) processor resources allocated to the CPC image object.</p> <p>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor. Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor.</p>
7F (127) HWI_IFLIPWCAP	<p>A 4-byte integer type value. This indicates whether the initial processing weight for integrated facility for Linux (IFL) processors is a limit or a target.</p> <p>HWMCA_TRUE Capped</p> <p>HWMCA_FALSE Not capped</p>

SetType values in hexadecimal, (decial), and equate symbol	Values to be specified
<p>80 (128) HWI_IFLPWMIN</p>	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the minimum relative amount of shared integrated facility for Linux (IFL) processor resources allocated to the CPC image object.</p> <p>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor.</p> <p>Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor.</p>
<p>81 (129) HWI_IFLPWMAX</p>	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the maximum relative amount of shared integrated facility for Linux (IFL) processor resources allocated to the CPC image object.</p> <p>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor.</p> <p>Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated facility for Linux (IFL) processor.</p>
<p>84 (132) HWI_ICFIPW</p>	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the relative amount of shared internal coupling facility (ICF) processor resources allocated to the CPC image object.</p> <p>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor.</p> <p>Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor.</p>
<p>85 (133) HWI_ICFIPWCAP</p>	<p>A 4-byte integer type value. This indicates whether the initial processing weight for internal coupling facility (ICF) processors is a limit or a target.</p> <p>HWI_ICFIPWCAP_TRUE Capped</p> <p>HWI_ICFIPWCAP_FALSE Not capped</p>

HWISET

SetType values in hexadecimal, (decial), and equate symbol	Values to be specified
86 (134) HWI_ICFPWMIN	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the minimum relative amount of shared internal coupling facility (ICF) processor resources allocated to the CPC image object.</p> <p>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor.</p> <p>Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor.</p>
87 (135) HWI_ICFPWMAX	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the maximum relative amount of shared internal coupling facility (ICF) processor resources allocated to the CPC image object.</p> <p>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor.</p> <p>Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated internal coupling facility (ICF) processor.</p>
8A (138) HWI_IIPW	<p>A 4-byte integer type value.</p> <p>A value from 1 - 999 defines the relative amount of shared integrated information processors (IIP) resources allocated to the CPC image object.</p> <p>A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated information processor (IIP).</p> <p>Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated information processor (IIP).</p>
8B (139) HWI_IIPWCAP	<p>A 4-byte integer type value. This indicates whether the initial processing weight for integrated information processors (IIP) is a limit or a target.</p> <p>HWMCAP_TRUE Capped</p> <p>HWMCAP_FALSE Not capped</p>

SetType values in hexadecimal, (decial), and equate symbol	Values to be specified
8C (140) HWI_IIPPWMIN	A 4-byte integer type value. A value from 1 - 999 defines the minimum relative amount of shared integrated information processors (IIP) resources allocated to the CPC image object. A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated information processor (IIP). Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated information processor (IIP).
8D (141) HWI_IIPPWMAX	A 4-byte integer type value. A value from 1 - 999 defines the maximum relative amount of shared integrated information processors (IIP) resources allocated to the CPC image object. A value of 0 indicates that CPC image does not represent a logical partition or the CPC image does not represent a logical partition with at least one not dedicated integrated information processor (IIP). Note: The setting of this attribute is only valid for CPC image objects that represent a logical partition with at least one not dedicated integrated information processor (IIP).
92 (146) HWI_GROUP_PROFILE_CAPACITY	A 4-byte integer value to represent the workload unit capacity for the group profile associated with an image.
C9 (201) HWI_IOCDS	A character string representing the IOCDS. A value of an empty string indicates that the reset activation profile will use the currently active IOCDS.
CA (202) HWI_IPL_ADDRESS	A character string representing the IPL address. Note: A value of an empty string indicates that the image activation profile uses the next IPL address set by HCD.
CB (203) HWI_IPL_PARM	A character string representing the IPL parameter. Note: A value of an empty string indicates that the image activation profile uses the next IPL parameter set by HCD.

HWISET

SetType values in hexadecimal, (decial), and equate symbol	Values to be specified
CC (204) HWI_IPL_TYPE	A 4-byte integer type value. HWMCA_IPLTYPE_STANDARD Indicates that the image activation profile is used to perform a standard load. HWMCA_IPLTYPE_SCSI Indicates that the image activation profile is used to perform a SCSI load. HWMCA_IPLTYPE_SCSIDUMP Indicates that the image activation profile is used to perform a SCSI dump.
CD (205) HWI_WW_PORTNAME	A character string representing the worldwide port name.
CE (206) HWI_BOOT_PGM_SELECTOR	A 4-byte integer type value representing the boot program selector value.
CF (207) HWI_LU_NUM	A character string representing the logical unit number.
D0 (208) HWI_BOOTREC_BLK_ADDR	A character string representing the boot record logical block address.
D1 (209) HWI_OPSYS_LOADPARM	A character string representing the operating system specific load parameters.
D2 (210) HWI_GROUP_PROF_NAME	A character string that represents the name of a group capacity profile.
D3 (211) HWI_LOAD_AT_ACTIVATION	A 4-byte integer type value. This indicates whether a load should be done at the end of activation. <ul style="list-style-type: none"> • HWMCA_TRUE • HWMCA_FALSE
D4 (212) HWI_CENTRAL_STOR	A 4-byte integer type value to represent the initial amount of central storage (in megabytes) to be used for the CPC image.

SetType values in hexadecimal, (decial), and equate symbol	Values to be specified
D5 (213) HWI_RES_CENTRAL_STOR	A 4-byte integer type value to represent the reserved amount of central storage (in megabytes) to be used for the CPC image.
D6 (214) HWI_EXPANDED_STOR	A 4-byte integer type value to represent the initial amount of expanded storage (in megabytes) to be used for the CPC image.
D7 (215) HWI_RES_EXPANDED_STOR	A 4-byte integer type value to represent the reserved amount of expanded storage (in megabytes) to be used for the CPC image.
D8 (216) HWI_NUM_GPP	A 4-byte integer type value to represent the number of dedicated general purpose processors to be used for the CPC image.
D9 (217) HWI_NUM_RESGPP	A 4-byte integer type value to represent the number of reserved dedicated general purpose processors to be used for the CPC image.
DA (218) HWI_NUM_IFA	A 4-byte integer value to represent the number of dedicated integrated facility for applications (IFA) processors to be used for the CPC image.
DB (219) HWI_NUM_RESIFA	A 4-byte integer value to represent the number of reserved dedicated integrated facility for applications (IFA) processors to be used for the CPC image.
DC (220) HWI_NUM_IFL	A 4-byte integer value to represent the number of dedicated integrated facility for Linux (IFL) processors to be used for the CPC image.
DD (221) HWI_NUM_RESIFL	A 4-byte integer value to represent the number of reserved dedicated integrated facility for Linux (IFL) processors to be used for the CPC image.
DE (222) HWI_NUM_ICF	A 4-byte integer value to represent the number of dedicated internal coupling facility (ICF) processors to be used for the CPC image.
DF (223) HWI_NUM_RESICF	A 4-byte integer value to represent the number of reserved dedicated internal coupling facility (ICF) processors to be used for the CPC image.

HWISET

SetType values in hexadecimal, (decial), and equate symbol	Values to be specified
E0 (224) HWI_NUM_ZIIP	A 4-byte integer value to represent the number of dedicated System z Integrated Information Processors (zIIPs) to be used for the CPC image.
E1 (225) HWI_NUM_RESZIIP	A 4-byte integer value to represent the number of reserved dedicated System z Integrated Information Processors (zIIPs) to be used for the CPC image.
E2 (226) HWI_NUM_SHARED_GPP	A 4-byte integer type value to represent the number of shared general purpose processors to be used for the CPC image.
E3 (227) HWI_NUM_RES_SHARED_GPP	A 4-byte integer type value to represent the number of reserved shared general purpose processors to be used for the CPC image.
E4 (228) HWI_NUM_SHARED_IFA	A 4-byte integer value to represent the number of shared integrated facility for applications (IFA) processors to be used for the CPC image.
E5 (229) HWI_NUM_RES_SHARED_IFA	A 4-byte integer value to represent the number of reserved shared integrated facility for applications (IFA) processors to be used for the CPC image.
E6 (230) HWI_NUM_SHARED_IFL	A 4-byte integer value to represent the number of shared integrated facility for Linux (IFL) processors to be used for the CPC image.
E7 (231) HWI_NUM_RES_SHARED_IFL	A 4-byte integer value to represent the number of reserved shared integrated facility for Linux (IFL) processors to be used for the CPC image.
E8 (232) HWI_NUM_SHARED_ICF	A 4-byte integer value to represent the number of shared internal coupling facility (ICF) processors to be used for the CPC image.
E9 (233) HWI_NUM_RES_SHARED_ICF	A 4-byte integer value to represent the number of reserved shared internal coupling facility (ICF) processors to be used for the CPC image.
EA (234) HWI_NUM_SHARED_ZIIP	A 4-byte integer value to represent the number of shared System z Integrated Information Processors (zIIPs) to be used for the CPC image.

SetType values in hexadecimal, (decmlal), and equate symbol	Values to be specified
EB (235) HWI_NUM_RES_SHARED_ZIIP	A 4-byte integer value to represent the number of reserved shared System z Integrated Information Processors (zIIPs) to be used for the CPC image.
EC (236) HWI_BASIC_CPU_AUTH_COUNT_CNTL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE The authorization control is enabled. HWMCA_FALSE The authorization control is disabled.
ED (237) HWI_PROBSTATE_CPU_AUTH_COUNT_CNTL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE The authorization control is enabled. HWMCA_FALSE The authorization control is disabled.
EE (238) HWI_CRYPTACTIVITY_CPU_AUTH_COUNT_CNTL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE The authorization control is enabled. HWMCA_FALSE The authorization control is disabled.
EF (239) HWI_EXTENDED_CPU_AUTH_COUNT_CNTL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE The authorization control is enabled. HWMCA_FALSE The authorization control is disabled.
F0 (240) HWI_COPROCESSOR_CPU_AUTH_COUNT_CNTL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE The authorization control is enabled. HWMCA_FALSE The authorization control is disabled.
F1 (241) HWI_BASIC_CPU_SAMPLING_AUTH_CNTL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE The authorization control is enabled. HWMCA_FALSE The authorization control is disabled.

HWISET

SetType values in hexadecimal, (decial), and equate symbol	Values to be specified
F2 (242) HWI_APROF_STORE_STATUS	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE Store status is selected. Only allowed if HWI_APROF_LOADTYPE is set to HWMCA_LOADTYPE_NORMAL. HWMCA_FALSE Store status is not selected.
F3 (243) HWI_APROF_LOADTYPE	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_LOADTYPE_NORMAL The Loadtype is set to normal. HWMCA_LOADTYPE_CLEAR The Loadtype is set to clear.
F4 (244) HWI_PROFILE_DESCRIPTION	A 50-character activation profile description. This attribute is only available when targeting a z10 or higher CPC.
F5 (245) HWI_PROFILE_PARTITION_ID	A 4-byte integer type decimal value ranging from 0 to 63. This attribute is only available when targeting a z10 or higher CPC.
F6 (246) HWI_OPERATING_MODE	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. <ul style="list-style-type: none"> • HWMCA_ESA390_OPERATING_MODE • HWMCA_ESA390TPF_OPERATING_MODE • HWMCA_CF_OPERATING_MODE • HWMCA_LINUX_OPERATING_MODE • HWMCA_FMEX_OPERATING_MODE • HWMCA_HMEX_OPERATING_MODE • HWMCA_HMAS_OPERATING_MODE • HWMCA_ZVM_OPERATING_MODE
F7 (247) HWI_CLOCK_TYPE	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. <ul style="list-style-type: none"> • HWMCA_CLOCK_TYPE_STANDARD • HWMCA_CLOCK_TYPE_LPAR
F8 (248) HWI_TIME_OFFSET_DAYS	A 4-byte integer type decimal value ranging from 0 - 999. This attribute is only available when targeting a z10 or higher CPC.
F9 (249) HWI_TIME_OFFSET_HOURS	A 4-byte integer type decimal value ranging from 0 - 23. This attribute is only available when targeting a z10 or higher CPC.

SetType values in hexadecimal, (decial), and equate symbol	Values to be specified
FA (250) HWI_TIME_OFFSET_MINUTES	A 4-byte integer type decimal value. Possible values are 0, 15, 30 or 45. This attribute is only available when targeting a z10 or higher CPC.
FB (251) HWI_TIME_OFFSET_INCREASE	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE The local time zone is east of GMT. HWMCA_FALSE The local time zone is west of GMT.
FC (252) HWI_LICCC_VALIDATION_ENABLED	A 4-byte integer type value. This attribute is only available when targeting a zEnterprise or higher CPC. HWMCA_TRUE Activation profile must conform to the current LICCC configuration. HWMCA_FALSE Activation profile is not required to conform to the current LICCC configuration.
FD (253) HWI_GLOBAL_PERFORMANCE _DATA_CONTROL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE Global performance data control is enabled. HWMCA_FALSE Global performance data control is disabled.
FE (254) HWI_IO_CONFIGURATION _CONTROL	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE I/O configuration control is enabled. HWMCA_FALSE I/O configuration control is disabled.
100 (256) HWI_LOGICAL_PARTITION _ISOLATION	A 4-byte integer type value. This attribute is only available when targeting a z10 or higher CPC. HWMCA_TRUE Logical partition isolation control is enabled. HWMCA_FALSE Logical partition isolation control is disabled.

SetTypeValueLen (non-REXX)

Supplied parameter

- Type: Integer
- Length: 4 bytes

SetTypeValueLen specifies the length in bytes of the **SetTypeValue** pointed to by the **SetTypeValue_Ptr** parameter. Some **SetType** requests allow a null value to be set. If a null value is desired, the **SetTypeValueLen** value must be zero.

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter

- Type: Character string (non-REXX), stem variable (REXX)
- Length: 32 bytes (non-REXX)

DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the **DiagArea** can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the **DiagArea** might not be filled in, and the data returned in the area should be ignored.

Field Name (non-REXX) / Tail name constant of the user-defined DiagArea stem (REXX)	Field Type (non-REXX)	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code that is returned from the Console Application API or the BCPii transport layer.
Diag_Text	Character (12)	Reserved.

See Appendix A, “BCPii communication error reason codes,” on page 583 for a partial list of the descriptive communication transport error return codes and suggested actions.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0007yyyy' because of one of the following reasons:

Table 67. Reasons for abend X'042', RC X'0007yyyy'

yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See *z/OS MVS System Codes* for additional information.

Return codes

When the service returns control to the caller, GPR 15 and **ReturnCode** contain a hexadecimal return code.

Hexadecimal return code Equate symbol	Meaning and action
0 HWI_OK	<p>Meaning: Successful completion. An SMF record has been written.</p> <p>Action: None.</p>
100 HWI_CONNECT_TOKEN_INV	<p>Meaning: Program error. The specified connect token is not valid. This return code indicates one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The connect token does not exist. A previous HWICONN service call has never returned the value specified on OutConnectToken. • The connect token does not represent an active connection. The connection specified might have already been disconnected using the HWIDISC service call. • The connect token is not associated with the address space of the caller. The ConnectToken specified is associated with a different address space than the caller of this service call. <p>Action: Check for probable coding error.</p>
101 HWI_COMMUNICATION_ERROR	<p>Meaning: A communication error is detected. The hardware management console application API (HWMCA) or the BCPii Transport layer has returned with a failing return code.</p> <p>Action: Check for probable coding error. See the DiagArea for further diagnostic information. The Diag_CommErr indicates the return code that is returned from HWMCA APIs or the BCPii Transport layer.</p> <p>HWMCA API and BCPii transport return codes are provided in Appendix A, "BCPii communication error reason codes," on page 583.</p>
102 HWI_DIAGAREA_INV	<p>Meaning: Program error. The DiagArea is not accessible.</p> <p>Action: Check for probable coding error. Verify the specified DiagArea is defined as a 32-byte character field.</p>

|
|

Hexadecimal return code Equate symbol	Meaning and action
103 HWI_CONNECT_TOKEN_INACTIVE	<p>Meaning: The specified connect token is no longer valid. The connection has been disconnected or it is in the progress of being disconnected.</p> <p>Action: Check for probable coding error. Verify that the specified connect token is still active. If connectivity to the targeted CPC connection no longer exists, all connections associated with that CPC will no longer have a connect token that can be used.</p>
501 HWI_SETTYPE_INV	<p>Meaning: Program error. The requested SetType specified in the call is not valid for the ConnectToken specified. The system rejects the service call. This return code indicates one of the following conditions has occurred:</p> <ul style="list-style-type: none"> • The SetType specified is not in the acceptable value range of attributes that can be set. • The specified SetType has been provided with an incompatible connection type. For example, the attribute identifier applies only to CPC connections, but the ConnectToken specified represents an image connection, or any of the activation profile connections. <p>Action: Check for probable coding error. Validate that the SetType specified is in the valid range of possible values. Validate that the SetType specified is permitted for the specified connection type.</p> <p>See the DiagArea for further diagnostic information.</p> <ul style="list-style-type: none"> • The Diag_Key contains the value of the attribute in question. • The Diag_Text contain “Bad Set Attr” if the value of the attribute cannot be set; the Diag_Text contains “Mismatch” if the attribute cannot be set for the specified connection type.
502 HWI_SETTYPE_VALUE_INV	<p>Meaning: Program error. The requested SetTypeValue to be set or changed is not valid. The system rejects the service call.</p> <p>Action: Check for probable coding error. Validate that the value to which an attribute is being set is appropriate for that attribute.</p>

Hexadecimal return code Equate symbol	Meaning and action
503 HWI_SETTYPE_VALUE_LEN_INV	<p>Meaning: Program error. The SetTypeValueLen specified is not valid. The SetTypeValueLen must be equal to or greater than the minimum required length for the set type value.</p> <p>Action: Check for probable coding error. Validate that the SetTypeValueLen specified is equal to or greater than the minimum required length for the set type value.</p> <p>Note: If the application is setting the value to null for a request that allows a null SetTypeValue, ensure that both the SetTypeValueLen and SetTypeValue_Ptr parameters are set to zero.</p>
504 HWI_SETTYPE_VALUE_INACCESSIBLE	<p>Meaning: Program error. The set type value data area is either partially or completely inaccessible by the application, or BCPii, or both.</p> <p>Action: Check for probable coding error. Verify the SetTypeValue_Ptr points to a data area where the set type value is, and make sure that the data area is accessible.</p>
506 HWI_SET_ATTRIBUTE_NOT_SUPPORTED	<p>Meaning: The targeted hardware of the HWISET request does not recognize the attribute that the user is attempting to set.</p> <p>Action: Verify that the targeted hardware is at a level that supports the type of attribute being set.</p>
F00 HWI_NOT_AVAILABLE	<p>Meaning: HWI is not available, and the system rejects the service request.</p> <p>Action: Start HWI and try the request again.</p>
F01 HWI_AUTH_FAILURE	<p>Meaning: The caller is PKM8-15 problem state.</p> <p>Action: Check the calling program for a probable coding error.</p>

|
|
|
|
|

Hexadecimal return code Equate symbol	Meaning and action
F02 HWI_NO_SAF_AUTH	<p>Meaning: The user does not have correct SAF authorization for the request.</p> <p>Action: Check for probable error. Consider one or more of the following possible actions:</p> <ul style="list-style-type: none"> • Define read access authorization to the FACILITY class resource profile HWI.APPLNAME.HWISERV. • Define update access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau for a CPC connection. • Define update access authorization to the FACILITY class resource profile HWI.TARGET.netid.nau.imagename for an image connection. • Ensure that the referenced Facility Class Profile is RACLIST-specified.
F03 HWI_INTERRUPT_STATUS_INV	<p>Meaning: The calling program is disabled. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWI_MODE_INV	<p>Meaning: The calling program is not in Task mode, which is the required mode. The system rejects this service request.</p> <p>Action: Check the calling program for a probable error.</p>
F05 HWI_LOCKS_HELD	<p>Meaning: The calling program is holding one or more locks. The system rejects this service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F06 HWI_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects this service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports HWI. Then rerun the calling program.</p>
F07 HWI_UNSUPPORTED_ENVIRONMENT	<p>Meaning: The system does not support execution of the service from the current environment (for example, calling a BCPii service from within a BCPii ENF exit routine).</p> <p>Action: Issue the BCPii service from a different execution environment.</p>

Hexadecimal return code Equate symbol	Meaning and action
FFF HWI_UNEXPECTED_ERROR	<p>Meaning: System error. The service that was called encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center. In many cases, a dump has been taken by BCPii to attempt the collection of the necessary information to diagnose the error. If so, provide this dump to the IBM support team.</p>

Example

In the pseudocode example, the caller issues a call to change or set the CPC status for a CPC.

```
.
.
SetType = HWI_ACCSTAT;
SetTypeValue = HWMCA_STATUS_OPERATING;
SetTypeValue_Ptr = addr(SetTypeValue);
SetTypeValueLen = Length(SetTypeValue);
CALL HWISET (ReturnCode, ConnectToken, SetType, SetTypeValue_Ptr,
             SetTypeValueLen, DiagArea)
```

A REXX programming example for the HWISET service:

```
mySetType = HWI_ACCSTAT           /* AccStat attribute       */
mySetTypeValue = HWMCA_STATUS_EXCEPTIONS

address bcpii
             "hwiset RetCode myConnectToken mySetType mySetTypeValue myDiag."
```

```
If (RC <> 0) | (Retcode <> 0) Then
  Do
    Say 'Service failed with REXX RC = 'RC' and API Retcode = 'Retcode'.'
    If (RC=Hwi_REXXParmSyntaxError | Retcode<>0) Then
      Do
        Say ' Diag_index=' myDiag.DIAG_INDEX
        Say ' Diag_key=' myDiag.DIAG_KEY
        Say ' Diag_actual=' myDiag.DIAG_ACTUAL
        Say ' Diag_expected=' myDiag.DIAG_EXPECTED
        Say ' Diag_commerr=' myDiag.DIAG_COMMERR
        Say ' Diag_text=' myDiag.DIAG_TEXT
      End
    End
```

HWIBeginEventDelivery — Begin delivery of BCPii event notifications

Call the HWIBeginEventDelivery service to allow a C application running in the z/OS UNIX System Services environment to begin delivery of event notifications. This service must be issued before the HWIManageEvents service.

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	None
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 and addressable by the caller
Linkage:	Standard C linkage conventions are used

Programming requirements

The file hwicmuss.x contains the sidedeck needed to link the program to the DLL.

z/OS UNIX C language callers must include the header file HWICIC.

Restrictions

None.

Authorization

Read access to the SAF profile CEA.CONNECT in the SERVAUTH class is required.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters in the order shown.

CALL statement	Parameters
int HWIBeginEventDelivery	(*DiagArea ,ConnectToken ,**DeliveryToken)

Parameters

The parameters are explained as follows:

*DiagArea

Returned parameter

- Type: character string
- Length: 32 bytes

*DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the *DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the *DiagArea might not be filled in, and the data returned in the area should be ignored.

Field Name	Field Type	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code from the failing operation.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, “BCPii communication error reason codes,” on page 583 for a partial list of the descriptive communication transport error return codes and suggested actions.

ConnectToken

Supplied parameter

- Type: character string
- Length: 16 bytes

ConnectToken specifies the value returned from an HWICONN service call.

**DeliveryToken

Returned parameter

- Type: character string
- Length: 8 bytes

**DeliveryToken specifies the variable to contain the address of the token that represents the event notification connection on future service calls.

ABEND codes

None.

Return codes

When the service completes, one of the following values is returned to the caller:

Return Code in Hexadecimal Equate Symbol	Meaning and Action
00000000 HWIUSS_RC_OK	<p>Meaning: Successful completion.</p> <p>Action: None.</p>

HWIBeginEventDelivery

Return Code in Hexadecimal Equate Symbol	Meaning and Action
00001001 HWIUSS_RC_UNAVAILABLE	<p>Meaning: This error is returned for one of the following reasons, which is written to the diag_commmerr field of the DiagArea:</p> <ul style="list-style-type: none"> • CEA (Common Event Adapter) communication is unavailable. (reason x'100') • Write access to a socket is denied. (reason x'103') • Services are failing in the CEA Server. (reason x'111') <p>Action: The request is rejected. Confirm that the CEA address space has been started and try the request again.</p>
00001002 HWIUSS_RC_NO_AUTH	<p>Meaning: The program is not authorized to access CEA services.</p> <p>Action: The request is rejected. Determine if the program needs access to CEA services. If so, grant the required access to the proper resources and try this request again. See "Setting up event notification for BCPii z/OS UNIX applications" on page 266 for further information.</p>
00001003 HWIUSS_RC_MAX_CLIENTS	<p>Meaning: The maximum number of CEA clients has been reached.</p> <p>Action: The request is rejected. Determine if other CEA clients can be stopped. If so, try this request again.</p>
00001007 HWIUSS_RC_SAF_NOTDEF_CONNECT	<p>Meaning: The SAF profile CEA.CONNECT is not defined.</p> <p>Action: The request is rejected. Add the CEA.CONNECT profile to the SERVAUTH class and try this request again.</p>
00001008 HWIUSS_RC_COMM_FAILURE	<p>Meaning: An error occurred in z/OS UNIX socket processing.</p> <p>Action: The request is rejected. Verify that the file system is properly configured for z/OS UNIX sockets and try this request again.</p>
00001009 HWIUSS_RC_CEA_INTERNAL_ERROR	<p>Meaning: An internal CEA processing error has occurred.</p> <p>Action: The request is rejected. Consult the DiagArea for the details about this error. If the error persists, contact the IBM Support Center.</p>
0000100A HWIUSS_RC_INPUT_PTR_IS_NULL	<p>Meaning: A null input pointer was found.</p> <p>Action: The request is rejected. Pass a valid pointer to the API and try this request again.</p>

Return Code in Hexadecimal Equate Symbol	Meaning and Action
0FFFFFFF HWIUSS_RC_UNEXPECTED_ERROR	<p>Meaning: An unexpected error has occurred.</p> <p>Action: The request is rejected. Consult the DiagArea for more specifics regarding the error. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Example

In the C code example, the caller issues a call to register for event delivery.

```
HWI_CONNTOKEN_TYPE hwitoken;
HWI_DIAGAREA_TYPE DiagArea;
HWI_DELIVERYTOKEN_TYPE *DeliveryToken;
int localRC;
```

```
localRC = HWIBeginEventDelivery(&DiagArea, hwitoken, DeliveryToken)
```

HWIEndEventDelivery — End delivery of BCPii event notifications

Call the HWIEndEventDelivery service to allow a C application running in the z/OS UNIX System Services environment to end delivery of event notifications. This service unregisters the registration made by the HWIBeginEventDelivery service.

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	None
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 and addressable by the caller
Linkage:	Standard C linkage conventions are used

Programming requirements

The file hwicmuss.x contains the sidedeck needed to link the program to the DLL.

z/OS UNIX C language callers must include the header file HWICIC.

Restrictions

None.

Authorization

Read access to the SAF profile CEA.CONNECT in the SERVAUTH class is required.

HWIEndEventDelivery

Syntax

Write the call as shown on the syntax diagram. You must code all parameters in the order shown.

CALL statement	Parameters
int HWIEndEventDelivery	(*DiagArea ,*DeliveryToken)

Parameters

The parameters are explained as follows:

*DiagArea

Returned parameter

- Type: character string
- Length: 32 bytes

*DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the *DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the *DiagArea might not be filled in, and the data returned in the area should be ignored.

Field Name	Field Type	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code from the failing operation.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, "BCPii communication error reason codes," on page 583 for a partial list of the descriptive communication transport error return codes and suggested actions.

*DeliveryToken

Supplied parameter

- Type: character string
- Length: 8 bytes

DeliveryToken specifies the event notification connection created by a previous HWIBeginEventDelivery call.

ABEND codes

None.

Return codes

When the service completes, one of the following values is returned to the caller:

Return Code in Hexadecimal Equate Symbol	Meaning and Action
00000000 HWIUSS_RC_OK	<p>Meaning: Successful completion.</p> <p>Action: None.</p>
00001001 HWIUSS_RC_UNAVAILABLE	<p>Meaning: This error is returned for one of the following reasons, which is written to the diag_commerc field of the DiagArea:</p> <ul style="list-style-type: none"> • CEA (Common Event Adapter) communication is unavailable. (reason x'100') • Write access to a socket is denied. (reason x'103') • Services are failing in the CEA Server. (reason x'111') <p>Action: The request is rejected. Confirm that the CEA address space has been started and try the request again.</p>
00001004 HWIUSS_RC_BAD_DELIVERYTOKEN	<p>Meaning: The provided delivery token is not valid.</p> <p>Action: The request is rejected. This is a probable coding error.</p>
00001008 HWIUSS_RC_COMM_FAILURE	<p>Meaning: An error occurred in z/OS UNIX socket processing.</p> <p>Action: The request is rejected. Verify that the file system is properly configured for z/OS UNIX sockets and try this request again.</p>
00001009 HWIUSS_RC_CEA_INTERNAL_ERROR	<p>Meaning: An internal CEA processing error has occurred.</p> <p>Action: The request is rejected. Consult the DiagArea for the details about this error. If the error persists, contact the IBM Support Center.</p>
0000100A HWIUSS_RC_INPUT_PTR_IS_NULL	<p>Meaning: A null input pointer was found.</p> <p>Action: The request is rejected. Pass a valid pointer to the API and try this request again.</p>
0FFFFFFF HWIUSS_RC_UNEXPECTED_ERROR	<p>Meaning: An unexpected error has occurred.</p> <p>Action: The request is rejected. Consult the DiagArea for more specifics regarding the error. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Example

In the C code example, the caller issues a call to unregister for event delivery.

HWIEndEventDelivery

```
HWI_DIAGAREA_TYPE DiagArea;  
HWI_DELIVERYTOKEN_TYPE *DeliveryToken;  
int localRC;  
  
localRC = HWIEndEventDelivery(&DiagArea, DeliveryToken)
```

HWIManageEvents — Manage the list of BCPii events

Call the HWIManageEvents service to allow a C application running in the z/OS UNIX System Services environment to manage the list of events for which the application is to be notified. The HWIBeginEventDelivery service must have been called before the HWIManageEvents service being called because the appropriate delivery token returned from the HWIBeginEventDelivery service is required as input.

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	One of the following: PKM allowing key 0-7, supervisor state, or APF-Authorized
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any 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 and addressable by the caller
Linkage:	Standard C linkage conventions are used

Programming requirements

The file hwicmuss.x contains the sidedeck needed to link the program to the DLL.

z/OS UNIX C language callers must include the header file HWICIC.

Restrictions

None.

Authorization

The client application must have access to consult the local CPC. This is granted by allowing the application at least read access to the SAF-protected FACILITY class resource HWI.APPLNAME.HWISERV.

Read access is required to the profile CEA.SUBSCRIBE.ENF_0068qqqqqqqq in the SERVAUTH class, where qqqqqqqq is the specific hexadecimal event qualifier pattern. See the ENF 68 documentation contained in the ENFREQ chapter of *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for further information about how to specify this event qualifier.

The client application must have at least read access to the SAF-protected FACILITY class resource HWI.TARGET.netid.nau for a ConnectToken representing a CPC connection, or HWI.TARGET.netid.nau.imagename for a ConnectToken representing an image connection.

Note: BCPii requires the FACILITY class to be RACLIST-specified.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters in the order shown.

CALL statement	Parameters
int HWIManageEvents	(*DiagArea ,*DeliveryToken ,ConnectToken ,EventAction ,EventIDs)

Parameters

The parameters are explained as follows:

*DiagArea

Returned parameter

- Type: character string
- Length: 32 bytes

*DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the *DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the *DiagArea might not be filled in, and the data returned in the area should be ignored.

Field Name	Field Type	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code from the failing operation.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, “BCPii communication error reason codes,” on page 583 for a partial list of the descriptive communication transport error return codes and suggested actions.

*DeliveryToken

Supplied parameter

- Type: character string
- Length: 8 bytes

HWIManageEvents

*DeliveryToken specifies the event notification connection, as returned by a previous HWIBeginEventDelivery call.

ConnectToken

Supplied parameter

- Type: character string
- Length: 16 bytes

ConnectToken specifies a logical connection between the application and a CPC or an image. The ConnectToken is an output parameter on the HWICONN service call.

The ConnectToken specified must have originated from a HWICONN service call that was issued from the same address space as this service call.

EventAction

Supplied parameter

- Type: integer
- Length: 4 bytes

EventAction specifies the type of action for the service. See the EventAction parameter of “HWIEVENT — Register or unregister for BCPii events” on page 317 for the exact syntax.

EventIDs

Supplied parameter

- Type: integer
- Length: 128 bit (16 bytes)

EventIDs specifies the events to be added or deleted. See the EventIDs parameter of “HWIEVENT — Register or unregister for BCPii events” on page 317 for the exact syntax.

IBM recommends that an application should at least add the Hwi_Event_BCPIIStatus event if other events are going to be added by the application. The only way to listen for BCPii events in the z/OS UNIX System Services environment is to issue a blocking call to the HwiGetEvent service. If BCPii stops and the Hwi_Event_BCPIIStatus has not been added, the application has no way of knowing of this termination and may hang indefinitely. By at least listening to this event, an application can be aware of BCPii terminations and take the appropriate action.

ABEND codes

If BCPii is unable to properly access the user-supplied parameter list, the call might result in an abend X'042' with a reason code of X'0004yyyy' because of one of the following reasons:

Table 68. Reasons for abend X'042', RC X'0004yyyy'

yyyy	Reason
0000	The parameters passed by the caller are not in the primary address space.
0001	The parameters passed by the caller are not accessible.
0002	The number of parameters passed by the caller is not correct.

For other severe BCPii errors encountered during the call, an abend X'042' with a different reason code may result. See *z/OS MVS System Codes* for additional information.

Return codes

When the service completes, one of the following values is returned to the caller:

Return Code in Hexadecimal Equate Symbol	Meaning and Action
00000000 HWIUSS_RC_OK	<p>Meaning: Successful completion.</p> <p>Action: None.</p>
00001000 HWIUSS_RC_HWIEVENT_FAILURE	<p>Meaning: The resultant HWIEVENT service call failed.</p> <p>Action: The request is rejected. The DiagArea contains the failure data. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
00001001 HWIUSS_RC_UNAVAILABLE	<p>Meaning: This error is returned for one of the following reasons, which is written to the diag_commerr field of the DiagArea:</p> <ul style="list-style-type: none"> • CEA (Common Event Adapter) communication is unavailable. (reason x'100') • Write access to a socket is denied. (reason x'103') • Services are failing in the CEA Server. (reason x'111') <p>Action: The request is rejected. Confirm that the CEA address space has been started and try the request again.</p>
00001002 HWIUSS_RC_NO_AUTH	<p>Meaning: This error is returned for one of the following reasons, which is written to the diag_commerr field of the DiagArea:</p> <ul style="list-style-type: none"> • The program is not authorized to access CEA services. (reason x'102') • The program is not authorized to monitor the requested event. (reason x'10E') <p>Action: The request is rejected. Determine whether the program needs access to CEA services. If so, grant the required access to the proper resources and try this request again. See "Setting up event notification for BCPii z/OS UNIX applications" on page 266 for further information.</p>
00001004 HWIUSS_RC_BAD_DELIVERYTOKEN	<p>Meaning: The provided delivery token is not valid.</p> <p>Action: The request is rejected. This is a probable coding error.</p>
00001006 HWIUSS_RC_SAF_NOTDEF_EVENT	<p>Meaning: The SAF profile CEA.SUBSCRIBE.ENF_0068* is not defined.</p> <p>Action: The request is rejected. Add the proper CEA.SUBSCRIBE.ENF_0068* profile to the SERVAUTH class and try this request again.</p>

HWIManageEvents

Return Code in Hexadecimal Equate Symbol	Meaning and Action
00001008 HWIUSS_RC_COMM_FAILURE	<p>Meaning: An error occurred in z/OS UNIX socket processing.</p> <p>Action: The request is rejected. Verify that the file system is properly configured for z/OS UNIX sockets and try this request again.</p>
00001009 HWIUSS_RC_CEA_INTERNAL_ERROR	<p>Meaning: An internal CEA processing error has occurred.</p> <p>Action: The request is rejected. Consult the DiagArea for the details about this error. If the error persists, contact the IBM Support Center.</p>
0000100A HWIUSS_RC_INPUT_PTR_IS_NULL	<p>Meaning: A null input pointer was found.</p> <p>Action: The request is rejected. Pass a valid pointer to the API and try this request again.</p>
0FFFFFFF HWIUSS_RC_UNEXPECTED_ERROR	<p>Meaning: An unexpected error has occurred.</p> <p>Action: The request is rejected. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Example

In the C code example, the caller issues a call to register to be notified when the command response events and status change events occur.

```
HWI_DIAGAREA_TYPE DiagArea;
HWI_DELIVERYTOKEN_TYPE *DeliveryToken;
HWI_CONNTOKEN_TYPE ConnectToken;
HWI_EVENTIDS_TYPE EventIDs;
int localRC;

memset ((void*)&eventIDs, 0x00, sizeof (eventIDs));
memcpy (eventIDs.Hwi_EventID_EyeCatcher
        ,HWI_EVENTID_TEXT
        ,sizeof (eventIDs.Hwi_EventID_EyeCatcher));
EventIDs.Hwi_Event_CmdResp = 1;
EventIDs.Hwi_Event_StatusChg = 1;
localRC = HWIManageEvents(&DiagArea, DeliveryToken, ConnectToken,
                          HWI_EVENT_ADD, EventIDs)
```

HWIGetEvent — Retrieve outstanding BCPii event notifications

Call the HWIGetEvent service to allow a C application running in the z/OS UNIX System Services environment to retrieve outstanding BCPii event notifications.

Description

Environment

The requirements for the callers are:

Requirement	Details
Minimum authorization:	None
Dispatchable unit mode:	Task

Requirement	Details
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 and addressable by the caller
Linkage:	Standard C linkage conventions are used

Programming requirements

The file hwicmuss.x contains the sidedeck needed to link the program to the DLL.

z/OS UNIX C language callers must include the header file HWICIC.

Restrictions

None.

Authorization

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters in the order shown.

CALL statement int HWIGetEvent	Parameters (*DiagArea ,*DeliveryToken ,*Buffer ,*BufferSize ,*Timeout ,*BytesNeeded)
--	---

Parameters

The parameters are explained as follows:

*DiagArea

Returned parameter

- Type: character string
- Length: 32 bytes

*DiagArea contains diagnostic data to help determine the cause of a failure from the service. For many return codes, the *DiagArea can contain further information to help determine the cause of the failure. See the descriptions of different return codes for a partial list of data returned in this area.

Note: For all environmental errors (with return code X'F00' and higher), the *DiagArea might not be filled in, and the data returned in the area should be ignored.

HWIGetEvent

Field Name	Field Type	Description
Diag_Index	32-bit integer	The array index to the parameter field that causes the error.
Diag_Key	32-bit integer	The constant value represents the field that causes the error.
Diag_Actual	32-bit integer	The incorrect actual value specified.
Diag_Expected	32-bit integer	The expected value to be used.
Diag_CommErr	32-bit integer	The returned code from the failing operation.
Diag_Text	Character (12)	Additional diagnostic information in text format.

See Appendix A, “BCPii communication error reason codes,” on page 583 for a partial list of the descriptive communication transport error return codes and suggested actions.

*DeliveryToken

Supplied parameter

- Type: character string
- Length: 8 bytes

*DeliveryToken specifies the event notification connection, as returned by a previous HWIBeginEventDelivery call.

*Buffer

Supplied parameter

- Type: character string
- Length: up to 4096 bytes

*Buffer specifies the address of the storage where the ENF68 event data is to be returned. This data is mapped by the HWIENF68 structure in the HWICIC header file.

BufferSize

Supplied parameter

- Type: integer
- Length: 4 bytes

BufferSize specifies the size of the *Buffer storage area.

Constant HWIUSS_MAX_GETBUFFER_SIZE can be used to allocate a buffer large enough to hold the maximum size of ENF68 data returned.

Timeout

Supplied parameter

- Type: integer
- Length: 4 bytes

Timeout specifies the amount of time, in seconds, for which the service should wait for an event to occur.

Constant in Hexadecimal Equate Symbol	Description
0 HWIUSS_TIMEOUT_NOWAIT	Do not wait for an event to occur if one is not ready for delivery.
FFFFFFFF HWIUSS_TIMEOUT_INFINITE	Do not return until an event has occurred.
Any other non-negative number	Wait for the specified number of seconds.

Note: If the Hwi_Event_BCPIIStatus event is not registered by the application and the BCPii address space goes down, this service will not be completed if HWIUSS_TIMEOUT_INFINITE was specified. If a numeric value was specified, the service will wake up but neither event data nor indicator that BCPii is not available will be returned. IBM recommends that an application specifies the Hwi_Event_BCPIIStatus event on the HwiManageEvents service call if the HwiGetEvent service is used. When the HwiGetEvent service returns control to the application, an inspection of which event was received will allow the application to react appropriately when BCPii stops.

*BytesNeeded

Returned parameter

- Type: integer
- Length: 4 bytes

*BytesNeeded specifies the variable to contain the number of bytes used in the output buffer to contain the returned event data. If the buffer is not large enough to contain all the event data, this variable contains the amount of storage required to receive all the event data.

ABEND codes

None.

Return codes

When the service completes, one of the following values is returned to the caller:

Return Code in Hexadecimal Equate Symbol	Meaning and Action
00000000 HWIUSS_RC_OK	Meaning: Successful completion. Action: None.
00000001 HWIUSS_RC_PARTIAL_DATA	Meaning: The provided buffer was not large enough to contain all the event data. Action: The request is successful. To receive all the event data, buffer the size of which is at least BytesNeeded must be provided.
00000002 HWIUSS_RC_EVENTS_LOST	Meaning: At least one event was not returned because the program has not been retrieving events timely. Action: The request is successful. To receive all events, the program must make this service call more often or reduce the number of events requested.
00000003 HWIUSS_RC_TIMEOUT	Meaning: No events have occurred in the requested time interval. Action: The request is successful.

HWIGetEvent

Return Code in Hexadecimal Equate Symbol	Meaning and Action
00001001 HWIUSS_RC_UNAVAILABLE	<p>Meaning: This error is returned for one of the following reasons, which is written to the diag_commer field of the DiagArea:</p> <ul style="list-style-type: none"> • CEA (Common Event Adapter) communication is unavailable. (reason x'100') • Write access to a socket is denied. (reason x'103') • Services are failing in the CEA Server. (reason x'111') <p>Action: The request is rejected. Confirm that the CEA address space has been started and try the request again.</p>
00001004 HWIUSS_RC_BAD_DELIVERYTOKEN	<p>Meaning: The provided delivery token is not valid.</p> <p>Action: The request is rejected. This is a probable coding error.</p>
00001005 HWIUSS_RC_SMALL_BUFFER	<p>Meaning: The provided buffer is not large enough to contain the event data.</p> <p>Action: The request is rejected. This is a probable coding error. Provide a larger buffer and try the request again.</p>
00001008 HWIUSS_RC_COMM_FAILURE	<p>Meaning: An error occurred in z/OS UNIX socket processing.</p> <p>Action: The request is rejected. Verify that the file system is properly configured for z/OS UNIX sockets and try this request again.</p>
00001009 HWIUSS_RC_CEA_INTERNAL_ERROR	<p>Meaning: An internal CEA processing error has occurred.</p> <p>Action: The request is rejected. Consult the DiagArea for the details about this error. If the error persists, contact the IBM Support Center.</p>
0000100A HWIUSS_RC_INPUT_PTR_IS_NULL	<p>Meaning: A null input pointer was found.</p> <p>Action: The request is rejected. Pass a valid pointer to the API and try this request again.</p>
0FFFFFFF HWIUSS_RC_UNEXPECTED_ERROR	<p>Meaning: An unexpected error has occurred.</p> <p>Action: The request is rejected. Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Example

In the C code example, the caller issues a call to retrieve any outstanding event data, waiting forever until an event occurs.

```
HWI_DIAGAREA_TYPE DiagArea;
HWI_DELIVERYTOKEN_TYPE DeliveryToken;
char *Buffer[HWIUSS_MAX_GETBUFFER_SIZE];
int BufSize = HWIUSS_MAX_GETBUFFER_SIZE;
int Timeout = HWIUSS_TIMEOUT_INFINITE;
int BytesReturned;
int localRC;

localRC = HWIGetEvent(&DiagArea, DeliveryToken, &Buffer, BufSize,
                    Timeout, &BytesReturned)
```

HWIGetEvent

Part 9. z/OS client web enablement toolkit

The z/OS client web enablement toolkit provides a set of application programming interfaces (APIs) to enable traditional, native z/OS programs to participate in modern web services applications.

Introduction to the z/OS client web enablement toolkit

You can use web application APIs to create a client/server application using a request-response protocol that can link a client residing anywhere in the world with any web server. Many web applications have evolved to a simpler programming model based on representational state transfer (REST). Governed by a set of architectural constraints, RESTful applications can be much easier to develop, enabling the creation of elegant and secure web applications. RESTful applications typically use the ubiquitous Hypertext Transfer Protocol (HTTP) as the means of communication and either JavaScript Object Notation (JSON) or Extensible Markup Language (XML) as the format of data exchange between the client and server programs.

Applications running in traditional z/OS environments can play the client role of a RESTful web application and initiate a request to a web server residing on z/OS or any other platform that supports web applications. The z/OS client web enablement toolkit provides the following components to enable these applications to more easily participate in the client/server realm:

- A z/OS JSON parser to parse JSON text coming from any source, build new JSON, or add to existing JSON text, as described in Chapter 20, “The z/OS JSON parser,” on page 419
- A z/OS HTTP/HTTPS protocol enabler that uses interfaces similar to other industry-standard APIs, as described in Chapter 21, “The z/OS HTTP/HTTPS protocol enabler,” on page 497

While the primary focus of the toolkit is to enable traditional z/OS programs running in environments where these types of services are not as readily available (as compared to a z/OS UNIX or Java™ Virtual Machine (JVM) environment), the services can be run from virtually any environment on a z/OS system. Programs running as a batch job, as a started procedure, or running in almost any address space on a z/OS system can use the toolkit APIs in a similar manner to any standard z/OS APIs provided by the operating system. Furthermore, programs can invoke these APIs in the programming language of choice; the toolkit fully supports C/C++, COBOL, PL/I and high-level assembler languages.

Chapter 20. The z/OS JSON parser

The JSON parser portion of the z/OS client web enablement toolkit provides a generic, native z/OS JavaScript Object Notation (JSON) parser for z/OS applications.

JSON is a text-based open standard designed for human-readable data interchange. It is derived from the JavaScript scripting language for representing simple data structures and associative arrays, called objects. It is language-independent, with parsers available for many languages. JSON is described in RFC 7159 (<http://tools.ietf.org/html/rfc7159>). The official Internet media type for JSON is `application/json`. The JSON format is often used for serializing and transmitting structured data over a network connection. It is primarily used to transmit data between a server and web application, serving as an alternative to XML. JSON data representation is becoming more pervasive across the industry due to its simplicity and ease of use.

Both JSON and XML are recommended formats for the representation of a request/response body when programming an application following the principles of REST. The XML System Services component of z/OS (z/OS XML) provides a system-level XML parser that is integrated with the base z/OS operating system. It is intended for use by system components, middleware, and applications that need a simple, efficient XML parsing solution. Likewise, the z/OS JSON parser provided with the z/OS client web enablement toolkit is an equivalent system-level, general-purpose JSON parser that is integrated with the z/OS operating system and that works in any native z/OS environment.

JSON basics

The designation of JSON as a great data exchange format lies in its innate simplicity. There are only a few types of data (string, number, boolean, null, array, and object) and its data structures mirror many of the modern programming languages. Many resources are available for you to consult on the Internet, such as *Introducing JSON* (<http://json.org>), to help you gain a basic understanding of the easy-to-use syntax.

An important JSON concept to understand is the idea of an object entry. Within an object, there can be one or more unordered object entries. Each entry consists of a name/value pair. The name, represented by a string enclosed in double quotation marks, identifies the value portion of the pair. The value can be any valid JSON data type.

Figure 22 on page 420 shows an example of JSON text.

```

{
  "firstName": "Steve",
  "lastName": "Jones",
  "age": 46,
  "address": {
    "streetAddress": "123 Anywhere Ave",
    "city": "Poughkeepsie",
    "state": "NY",
    "postalCode": "12601",
    "country": "USA"
  },
  "phone": [
    {
      "type": "mobile",
      "number": "914 555 5555"
    },
    {
      "type": "home",
      "number": "845 555 1234"
    }
  ]
}

```

Figure 22. Example of JSON text

In Figure 22, "firstName", "lastName", "age", "address", and "phone" are all names of object entries in the main (root) object of the JSON text. The values assigned to the "firstName", "lastName", and "age" object entries are all simple data types (string, string, and number, respectively). The named entries "address" and "phone", however, contain a more complex data type as their values. The "address" entry nests additional address details within another object, which contains the portions of an address. The "phone" entry contains an array made up of two array entries. Array entries differ from object entries in that arrays contain only a value (of any JSON data type). In this case, the array values are two objects, each of which group specific types of phone numbers together.

Elements of the z/OS JSON parser

The z/OS JSON parser is organized into five types of services:

- **Initialize and terminate:** The purpose of these services is to prepare the memory space required by the z/OS JSON parser or to free it after parser services are no longer needed. The memory allocation created by the initialize service is known as a *parser instance*.
- **Parse:** The parse service performs the following functions:
 - Assigns a particular JSON text to a previously created parser instance
 - Checks the JSON text input for syntax errors
 - Creates an internal representation of the JSON text in the parser instance memory, allowing subsequent parse functions to execute quickly
- **Traverse:** These services are designed for applications that might not know the content of the JSON text they are parsing. By using these services, a program can easily traverse the JSON text input, one construct at a time, to discover what data was passed to it. For example, by using a recursive programming methodology, a few simple routines can easily and efficiently traverse the entire stream of data. See “z/OS JSON parser programming examples” on page 426 for examples provided by the toolkit in SYS1.SAMPLIB.
- **Search:** A program might need to find a particular name in a name or value pair that identifies a particular entry of interest. By using the search service and

specifying the appropriate search scope, the handle of the matching entry is returned. The handle can then be used on subsequent services to get the information at that location.

- **Create and serialize:** Beyond just reading JSON text created elsewhere and parsing through it, a program might need to create new JSON text, add a JSON entry into existing JSON text, or insert JSON text from another source into the middle of existing JSON text. The create service makes it easy to perform these functions.

When the program has completed its creation activities, it can then use the serialize service to build the JSON text and place it into a buffer in preparation for sending it to some program over a network or saving it in a local data store.

The general usage of the z/OS JSON parser services in an application follows this order:

1. Create a parser instance and obtain the parser handle (HWTJINIT).
2. Associate JSON text with the parser instance and create an internal representation of the data for fast access by subsequent services (HWTJPARS), or create new JSON text from scratch (HWTJCREN).
3. Traverse to discover the contents of the JSON text, or search to find a particular name in a name or value pair (HWTJSRCH), or add or insert new entries into the existing JSON text (HWTJCREN), or some combination of all of these services.
4. Build the new JSON text if any text was added (HWTJSERI).
5. Free storage used by the parser services (HWTJTERM).

Availability of the z/OS JSON parser

The z/OS JSON parser contained the z/OS client web enablement toolkit is available to virtually any address space. The toolkit is enabled as part of z/OS initialization during IPL time. A message is written to the syslog, which regards the status of the toolkit. Success or failure of toolkit initialization can be found by locating any HWT-prefixed syslog messages, which are issued during IPL.

Syntax, linkage, and programming considerations

The z/OS JSON parser is available to almost any program running in any address space. Almost all z/OS execution environments are supported as well as a wide variety of programming languages.

Programming interface files provided by the JSON parser

Table 69 lists the programming interface files provided by the z/OS JSON parser.

Table 69. JSON parser programming interface.

Programming language	Programming interface file
C / C++	Include file HWTJIC provided in SYS1.SIEAHDRV.H
COBOL	Copybook file HWTJICOB provided in SYS1.MACLIB
PL/I	Include file HWTJIPLI provided in SYS1.MACLIB
Assembler	Include file HWTJIASM provided in SYS1.MACLIB
REXX	See “HWTCONST — Initialize predefined variables (REXX)” on page 428 on how to access all the toolkit constants in REXX

Calling formats

Table 70 lists specific calling formats for languages that can invoke the z/OS JSON parser callable services.

Table 70. Calling formats for the z/OS JSON parser callable services.

Programming language	Calling format
C / C++	<i>Parser_service_name (return_code,param1,param2,...)</i>
COBOL	CALL <i>Parser-service-name</i> USING <i>return_code,param1,param2,...</i>
PL/I	CALL <i>Parser_service_name (return_code,param1,param2,...)</i>
Assembler	CALL <i>Parser_service_name (return_code,param1,param2,...)</i> ,VLIST
REXX	ADDRESS HWTJSON <i>"Parser_service_name return_code parm1 parm2..."</i>

Linkage considerations

There are three ways for a compiled application to find the z/OS JSON parser callable services:

- **Linkage stub method**

(Recommended) Use the linkable stub routine HWTJCSS from SYS1.CSSLIB to link edit your object code. If you attempt to run the parser on a previous release of z/OS that does not support the z/OS JSON parser, this method results in the service call receiving a return code of X'F03' (HWTJ_UNSUPPORTED_RELEASE).

- **Load method**

Use the LOAD macro to find the address of the z/OS JSON parsing callable service at run time and then CALL the service. If you attempt to run the parser on a previous release of z/OS that does not support the z/OS JSON parser, this method results in the LOAD macro failing to find the requested service.

- **Direct linkage method**

Code the linkage to the z/OS parser services directly. This can be done if the program first confirms that the level of z/OS contains the toolkit. The following example shows the assembler linkage:

```
L    R14,CVTCVRT-CVT(R14,0)
L    R14,84(R14,0)
L    R15,4*HWT_SERV_XXXXX(R14,0)
LR   R14,R0
BR   R15
```

In the example, *xxxxx* represents the last five letters of the service you want to call. This requires that the HWTJKASM assembler macro be included. If you attempt to run the parser on a previous release of z/OS that does not support the z/OS JSON parser, this method results in the application receiving an abend X'019'.

Linkage considerations for high-level language programming

Callers must ensure that the proper linkage is made to the JSON parsing services. The supplied IDF files for the various high-level languages contain the necessary definitions that ensure that the parameter list passed to the JSON parser has the high-order bit turned on for the last parameter. For example, for C, the linkage must be specified as OS linkage, such as:

```
#pragma linkage(HWTJxxxx_CALLTYPE,OS)
```

For PL/I, the entry declaration should have the following options defined:
OPTIONS(LINKAGE(SYSTEM))

Linkage considerations for assembler language programming

Callers must also use the following linkage conventions:

- Register 1 must contain the address of a parameter list that is a list of consecutive words, each containing the address of a parameter to be passed. The last word in this list must have a 1 in the high-order (sign) bit.
- Register 13 must contain the address of an 18-word save area.
- Register 14 must contain the return address.
- Register 15 must contain the entry point address of the service being called.
- If the caller is running in AR ASC mode, access registers 1, 13, 14, and 15 must all be set to zero.
- On return from the service, general and access registers 2 - 14 are restored (registers 0, 1 and 15 are not restored).

General programming considerations

Codepage considerations

All input data into the JSON parser is assumed to be in EBCDIC encoding (codepage 1047) before making any z/OS JSON parsing service call. Any JSON text received by the application in another encoding format must first be converted to EBCDIC before it can be input into the parser via either the parse (HWTJPARS) or create (HWTJCREN) service calls.

Recovery considerations

The z/OS JSON parser runs in the address space of the application. In addition, all the storage needed by the parser is obtained in the application's address space. Because every application has its own programming environment, it is impossible for the parser to predict the recovery environment required by the application; therefore, the parser does not provide its own recovery. It is imperative that a robust application provide recovery to catch any abnormal ends to parser execution.

When the parser is attempting to access application-provided parameters and those parameters are either inaccessible, point to an inaccessible location, or specify a length that goes beyond the available storage that is obtained by the application, an ABEND occurs and the recovery of the application (if established) receives control. To allow for easier debugging of the problem, when the parser is about to access any application-specified values, the **returnCode** parameter is pre-filled with the HWTJ_INACCESSIBLE_PARM return code and the **diagArea reasonDesc** value is pre-filled with the specifics of which parameter the z/OS JSON parser is attempting to access. If the parser abnormally ends with an S0C4 ABEND code due to an inaccessible parameter, the recovery routine can consult the **returnCode** and **diagArea** values in the callers dynamic storage at the time of the ABEND and see which parameter the parser could not process.

Lastly, if the application catches any abnormal ends (abends) during the z/OS JSON parser execution, subsequent parser calls using the same parser handle can fail with an HWTJ_PARSERHANDLE_INUSE return code. See the action description for this return code for a list of options a program can take when encountering this condition.

REXX Programming Considerations

The toolkit provides a REXX host command environment, HWTJSON, to allow REXX applications to parse and create JSON strings, as well as search within a JSON string. REXX applications running in TSO/E, System REXX, z/OS UNIX, or ISV-provided REXX environments are supported.

- To initialize the HWTJSON host command environment in your REXX exec, it may be necessary to invoke the hwtcalls function at the beginning of your application: **call hwtcalls on**. After this invocation, both the ADDRESS HWTHTTP and ADDRESS HWTJSON host commands will direct API calls to the toolkit.
- To declare all toolkit constants in your REXX exec, use the HWTCONST service as documented below.

Note: There is no REXX IDF (include file) provided by the toolkit.

- The toolkit services allocate task associated resources, which are released at task termination and the termination API calls.
- Handles are not shared among multiple tasks, which can restrict some reentrant REXX environments.
- JSON parser handles can be updated by any of the JSON parser services. The content of these variables should not be modified in any way by the application.
- Verify that all variables have proper content and are exposed if set outside of procedures.
- Variable names specified on toolkit REXX service calls are limited to 40 characters in length.
- REXX does not have unlimited variable content size. In general, a single variable cannot contain more than 16 MB of content. This limits the amount of data that can be sent and received in the JSON parser. If the data required is greater than 16 MB for any of these cases, consider to use one of the high-level languages, which are supported by the toolkit (C/C++, COBOL, PL/I or Assembler).
- The built-in REXX RC variable contains the return code from the REXX HWTJSON host command. This return code indicates the toolkit's acceptance of the supplied REXX HWTJSON host command. The return codes returned in the RC variable are generally unique to the REXX environment. In contrast, the JSON service return code, the variable supplied on the service call itself, is only filled in if the RC variable has a value of HWTJ_OK (0) or HWT_REXXParmSyntaxError(1). Possible return codes returned by the toolkit in the RC variable are:

Host command return codes for the REXX environment:

Table 71. Host return codes for REXX

Host return code	Meaning and action
0	Meaning: REXX toolkit host command successful. Action: Consult the toolkit return code on the service call to determine the final result of the request.

Table 71. Host return codes for REXX (continued)

Host return code	Meaning and action
<p>1</p> <p>HWT_REXXParmSyntaxError</p>	<p>Meaning: REXX toolkit host command detects the parameter format is not in the proper form to be accepted.</p> <p>Action:Check for a probable coding error.</p> <ul style="list-style-type: none"> • See the return code on the toolkit service call to determine the reason for the syntax error. • See the REXX programming considerations of the toolkit service to see the exact calling specifications. • Compare the toolkit REXX service call attempted with service call examples in the supplied toolkit REXX programming sample found in SYS1.SAMPLIB. • See the DiagArea for further diagnostic information.
<p>2</p> <p>HWT_REXXUnsupportedService</p>	<p>Meaning: Program error. An unknown toolkit service name was specified on the toolkit REXX host command.</p> <p>Action:Check for a probable coding error. Specify a valid toolkit service name. For example, HWTJPARS.</p>
<p>3</p> <p>HWT_REXXInvalidNumOfParms</p>	<p>Meaning: Program error. The number of parameters specified on the toolkit REXX host command for the service name specified does not match the number of parameters expected.</p> <p>Action:Check for a probable coding error. See the REXX programming considerations of the toolkit service to see the exact calling specifications. Compare the toolkit REXX service call attempted with service call examples in the supplied toolkit REXX programming sample found in SYS1.SAMPLIB.</p>
<p>4</p> <p>HWT_REXXStemVarRequired</p>	<p>Meaning: Program error. The toolkit REXX service specified on the toolkit REXX host command is missing one or more required stem variables in the positional parameter list.</p> <p>Action:Check for a probable coding error. See the REXX programming considerations of the toolkit service to see the exact calling specifications. A stem variable parameter must specify a "." following the variable name. For example, "var.". Also, compare the toolkit REXX service call attempted with service call examples found in the supplied toolkit REXX programming sample found in SYS1.SAMPLIB.</p>
<p>5</p> <p>HWT_REXXParmNameTooLong</p>	<p>Meaning: Program error. One or more variables specified on the toolkit REXX service call on the toolkit REXX host command is greater than the toolkit maximum REXX variable length (40).</p> <p>Action: Check for a probable coding error. Reduce the variable name lengths on the toolkit REXX service call to be 40 characters or less in length</p>
<p>6</p> <p>HWT_REXXInvalidHostEnv</p>	<p>Meaning: System error. The toolkit detected an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Table 71. Host return codes for REXX (continued)

Host return code	Meaning and action
32 HWT_REXXUnexpectedError	<p>Meaning: System error. An unexpected error is detected. The system rejects the service call.</p> <p>Action: A symptom record has been written to LOGREC to record the problem. Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

z/OS JSON parser programming examples

The z/OS JSON parser provides a sample program in many of the supported programming languages to aid in the creation of applications that use the parser functions. Each sample contains examples of how to use almost all of the JSON parser services available in the toolkit. The samples are shipped in SYS1.SAMPLIB. Table 72 lists the sample files for each programming language.

Table 72. JSON parser programming sample files

Programming language	Name of sample in SYS1.SAMPLIB
C / C++	HWTJXC1
COBOL	HWTJXCB1
PL/I	HWTJXPI1
REXX	HWTJXR1

z/OS JSON parser callable services

The z/OS JSON parser callable services are grouped under the following categories.

Initialization and termination services

Initialization and termination services deal with the creation and termination of JSON parser instances.

The z/OS JSON parser callable services in this category are:

- “HWTJINIT — Initialize a parser instance” on page 470
- “HWTJTERM — Terminate a parser instance” on page 492

Parse service

The parse service loads a selected JSON text stream into a particular JSON parser instance.

The z/OS JSON parser callable service in this category is:

- “HWTJPARS — Parse a JSON string” on page 474

Traversal (auto-discovery) parsing services

The purpose of the traversal services is to traverse (discover) the contents of the supplied JSON text in a methodical manner. Typically, these services take a supplied parser handle and either an **entryValueHandle** or **objectValueHandle**

parameter (returned by previous invocations of various JSON parser API calls) that points to a specific location within the JSON text.

Note: Throughout the documentation of the z/OS JSON parsing services, the word *entry* is used in the context of a JSON entry. When referring to an object, an entry represents a JSON name or value pair, where the name is a string enclosed in double quotation marks and the value can be specified as any valid JSON type. When referring to an array, an entry represents a value. The value can be any valid JSON type. An object or array can have one or more JSON entries. Multiple entries are separated by commas.

The z/OS JSON parser callable services in this category are:

- “HWTJGAEN — Get array entry” on page 439
- “HWTJGBOV — Get boolean value” on page 443
- “HWTJGJST — Get JSON type” on page 447
- “HWTJGNUE — Get number of entries” on page 451
- “HWTJGNUV — Get number value (non-REXX)” on page 455
- “HWTJGOEN — Get object entry” on page 460
- “HWTJGVAL — Get value” on page 466

Search service

The search service searches for a particular “*name*” in the JSON text.

The z/OS JSON parser callable service in this category is:

- “HWTJSRCH — Search” on page 485

JSON text creation methods

All of the other methods deal with existing JSON text that an application can process and analyze. However, there is also a need to be able to build JSON text. There are two ways to create JSON text using the z/OS web enablement toolkit:

- Create JSON text “from scratch.”
- Insert additional JSON text at a particular insertion point.

The toolkit also allows for the JSON text to be added using two different methods:

- **Adding one entry at a time:** By using the provided input parameters, the create service creates syntactically valid JSON text.
- **Adding previously defined JSON text:** The create service first parses the supplied JSON text to verify that the inserted JSON text contains no syntax errors. Then, the insertion point is validated to ensure that the supplied JSON text can logically be inserted at that point. This merges two JSON text streams into one larger text stream.

Once all of the JSON text is created, it is necessary for the application to use the serialize service to build the new JSON text stream containing all of the existing and new JSON text now associated with the parser instance.

The z/OS JSON parser callable services in this category are:

- “HWTJCREN — Create JSON entry” on page 429
- “HWTJSERI — Serialize (build) JSON text” on page 479

HWTCNST — Initialize predefined variables (REXX)

Call the HWTCNST service to initialize predefined variables in the current REXX variable pool.

Description

This service sets the variables with names prefixed for HWTJ corresponding to the interface definition for the JSON toolkit. This service is helpful when using symbolic names in checking for specific return codes or when specifying constant values in the application. The variable **HWT_CONSTANTS** is set to a list of the interface variable names, which is useful on a procedure expose statement to make the variables visible to a procedure.

Note: This service also sets the variables for the z/OS HTTP Enabler (HWTH-prefixed) as well. If the REXX application utilizes both the HTTP and JSON parser portions of the toolkit, it is only necessary to call HWTCNST once to initialize all the variables.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

REXX parameters
<pre>address hwtjson "hwtconst", "ReturnCode", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Character string

Contains the return code from the service.

DiagArea.

Returned parameter.

- **Type:** Stem variable

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in "HWTCNST — Initialize predefined variables (REXX)."

Table 73. Return codes for the HWTCONST service.

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.

HWTJCREN — Create JSON entry

Call the HWTJCREN service to create or insert new JSON text.

Description

The HWTJCREN service creates or inserts new JSON text into the specified parser instance.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJCREN service

All information for the HWTJCREN service applies for REXX requests except:

- EntryName replaces EntryNameAddr and EntryNameLen
- EntryValue replaces EntryValueAddr and EntryValueLen

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

HWTJCREN

Non-REXX parameters	REXX parameters
CALL HWTJCREN(ReturnCode, ParserHandle, ObjectHandle, EntryNameAddr, EntryNameLen, EntryValueType, EntryValueAddr, EntryValueLen, NewEntryValueHandle, DiagArea);	address hwjson "hwtjcren", "ReturnCode", "ParserHandle", "ObjectHandle", "EntryName", "EntryValueType", "EntryValue", "NewEntryValueHandle", "DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

ObjectHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies an object handle representing a particular JSON object (object or array object) that indicates where the JSON text is to be added (that is, defines the insertion point). The **objectHandle** value is either zero for the root object or an entry value handle (**entryValueHandle**) whose JSON type is HWTJ_OBJECT_TYPE or HWTJ_ARRAY_TYPE.

Because JSON does not allow for the ordering of object entries, a new entry will always be added after the current last entry.

If a new JSON text stream is being built, you must specify an **objectHandle** of zero. This service detects that the root object does not exist and will automatically create it. This service adds an entry to the root object.

Note: In the case of a new JSON text stream, it is not possible for this service to create an object entry consisting of an empty name string ("": *value*), that is, an **entryNameAddr** and **entryNameLen** of zero. If you want to have an object entry with an empty name string as the first object entry, take the following actions:

1. Create the standalone root object ({}), by calling the HWTJCREN service and specifying an **entryNameAddr**, **entryNameValue**, **entryValueAddr**, and **entryValueLen** of zero, and an **entryValueType** of HWTJ_OBJECTVALUETYPE.
2. Create an entry within the root object by calling the HWTJCREN service again and specifying an **entryNameAddr** and **entryNameValue** of zero, the wanted **entryValueType**, and an appropriate value.

If you specify an **objectHandle** of zero and there is existing JSON text already parsed within the parser instance, the new text is added as an entry to the root object.

EntryName (REXX)

Supplied parameter.

- **Type:** Character string

Specifies the REXX variable, which contains the name for the entry to be added. Set this variable to a null string if **entryname** is not applicable for the request.

Note: The supplied entry name must contain EBCDIC encoded data (codepage 1047).

EntryNameAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of a buffer that contains the name string associated with the entry to be created. This parameter is required if the **objectHandle** parameter specifies a normal object (JSON type is HWJT_OBJECT_TYPE). This parameter must be zero if the **objectHandle** parameter specifies an array object (JSON type is HWJT_ARRAY_TYPE), since array entries have no name.

If the specified **objectHandle** parameter is zero (that is, adding text to the root object) and there is no existing JSON text already parsed within the parser instance, a new JSON text stream is built. In this case, **entryNameAddr** must be zero if the **entryValueType** is either HWTJ_JSONTEXTVALUETYPE or HWTJ_OBJECTVALUETYPE (create a null root object).

Note: The supplied entry name buffer must contain EBCDIC encoded data (codepage 1047).

EntryNameLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the "name" string to be added at the location specified by **entryNameAddr**. This parameter is required if the **objectHandle** parameter specifies a normal object (JSON type is HWJT_OBJECT_TYPE). This parameter must be zero if the **objectHandle** parameter specifies an array object (JSON type is HWJT_ARRAY_TYPE), since array entries have no name.

If the specified **objectHandle** parameter is zero (that is, adding text to the root object) and there is no existing JSON text already parsed within the parser instance, a new JSON text stream is built. In this case, **entryNameLen** must be zero if the **entryValueType** is either HWTJ_JSONTEXTVALUETYPE or HWTJ_OBJECTVALUETYPE (create a null root object).

EntryValueType

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a constant value that indicates the data type of the value to be added.
Valid values are:

HWTJ_OBJECTVALUETYPE

The value to be added is an object.

HWTJ_ARRAYVALUETYPE

The value to be added is an array.

HWTJ_STRINGVALUETYPE

The value to be added is a string.

HWTJ_NUMVALUETYPE

The value to be added is a number.

HWTJ_TRUEVALUETYPE

The value to be added is a boolean value of TRUE.

HWTJ_FALSEVALUETYPE

The value to be added is a boolean value of FALSE.

HWTJ_NULLVALUETYPE

The value to be added is a NULL type.

HWTJ_JSONTEXTVALUETYPE

The value to be added is another JSON text stream.

If **entryValueType** is HWTJ_STRINGVALUETYPE, HWTJ_NUMVALUETYPE, or HWTJ_JSONTEXTVALUETYPE, the **entryValueAddr** and **entryValueLen** parameters must specify the actual *value* in the "name":*value* pair. If the **entryValueType** is not one of these three values, **entryValueAddr** and **entryValueLen** must be set to zero.

EntryValue (REXX)

Supplied parameter.

- **Type:** Character string

Specifies the REXX variable, which contains the value of the entry to be added.
Set this variable to a null string if **entryvalue** is not applicable for the request.

EntryValueAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of a buffer that contains the value to be added. This is only valid if **entryValueType** is HWTJ_STRINGVALUETYPE, HWTJ_NUMVALUETYPE, or HWTJ_JSONTEXTVALUETYPE. For all other **entryValueType** values, specify zero.

Notes:

1. If the **entryValueType** is HWTJ_JSONTEXTVALUETYPE, the parser maintains a binding to the buffer specified by this parameter even after the call to the HWTJCREN service completes. The buffer must not be freed or reused until the caller has completed all parsing functions associated with this JSON text string.
2. All **entryValueType** values that require a value to be specified, including HWTJ_JSONTEXTVALUETYPE, require the specified entry value buffer to contain EBCDIC encoded data (codepage 1047).

EntryValueLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the value to be added at the location specified by the **entryValueAddr** parameter. This is only valid if **entryValueType** is HWTJ_STRINGVALUETYPE, HWTJ_NUMVALUETYPE, or HWTJ_JSONTEXTVALUETYPE. For all other **entryValueType** values, specify zero.

NewEntryValueHandle

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

The handle that represents the new object entry that was created.

Note: If **entryValueType** is either HWTJ_OBJECTVALUETYPE or HWTJ_ARRAYVALUETYPE, the returned handle should be treated as an object handle (**objectHandle**) on subsequent requests, since the entry value is an object.

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0001yyyy' for one of the following reasons:

yyyy Reason

- 0000 The parameters passed by the caller are not in the primary address space.
- 0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 74.

Table 74. Return codes for the HWTJCREN service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.

Table 74. Return codes for the HWTJCREN service (continued)

Hexadecimal return code Equate symbol	Meaning and action
101 HWTJ_PARSERHANDLE_INV	<p>Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service).</p> <p>Action: Check for a probable coding error.</p>
102 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. Only pass one of the following values:</p> <ul style="list-style-type: none"> • An object handle or entry value handle on the objOrEntryHandle parameter that was returned by a prior z/OS JSON parser service call. • A value of zero for the root object.

Table 74. Return codes for the HWTJCREN service (continued)

Hexadecimal return code Equate symbol	Meaning and action
105 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified objectHandle does not represent an object or array object (JSON type of HWTJ_OBJECT_TYPE or HWTJ_ARRAY_TYPE).</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an objectHandle value that represents an object handle or array object handle.</p>
108 HWTJ_WORKAREA_TOO_SMALL	<p>Meaning: Program error. The work area (which contains the internal representation of the entire JSON text, including the JSON constructs, which the HWTJCREN service attempted to add) is not large enough to satisfy this HWTJCREN request. The parser requires a work area that is larger than the maxParserWorkAreaSize value that was specified on the HWTJINIT service.</p> <p>Action: Check for a probable coding error. IBM recommends specifying a maxParserWorkAreaSize value of 0 (unlimited work area size available to the parser). If the application cannot specify this value, then modify the invocation of the HWTJINIT service to specify a larger maxParserWorkAreaSize value based on the recommendations given in the diagDesc section of the diagArea and in the description of the HWTJINIT maxParserWorkAreaSize parameter.</p>
109 HWTJ_PARSE_ERROR	<p>Meaning: Supplied JSON text error. The JSON text passed by the caller to the HWTJCREN service contains a syntax error. The specified entryValueAddr points to a buffer containing JSON text to be added to the parser instance (entryValueType is HWTJ_JSONTEXTVALUETYPE). However, the supplied JSON text has one or more syntax errors.</p> <p>Note: This error can also be generated if the specified JSONTextLen value is greater than the actual length of the JSON text and there are non-null characters after the JSON text.</p> <p>Action: Check the diagArea for a complete explanation of the error. The reasonCode portion of the diagArea pinpoints the reason for the parse failure, while the diagDesc portion points to the exact location in the supplied JSON text where the parser detected a JSON syntax error.</p>
10B HWTJ_CANNOT_OBTAIN_WORKAREA	<p>Meaning: System error. The Storage Obtain service could not obtain the work area storage as required by the z/OS JSON parser during the HWTJCREN service call.</p> <p>Action: Consult the diagArea for the Storage Obtain failure return code and additional information found in the diagDesc section. Check to see if there is sufficient memory available in order for the parser to obtain the necessary amount of work area. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Table 74. Return codes for the HWTJCREN service (continued)

Hexadecimal return code Equate symbol	Meaning and action
201 HWTJ_JCREN_ENTRYNAMEADDR_INV	<p>Meaning: Program error. The caller specified a value of zero for the address of the entry name buffer when the specified objectHandle was of type HWTJ_OBJECT_TYPE or specified a nonzero value when the specified objectHandle was of type HWTJ_ARRAY_TYPE.</p> <p>Action: Check for a probable coding error. Specify the actual address of the buffer containing the entry name to be added if the objectHandle is of type HWTJ_OBJECT_TYPE. Specify zero if the objectHandle is of type HWTJ_ARRAY_TYPE.</p> <p>Note: Specifying a bad entry name buffer address other than zero can result in the parser terminating with an X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
202 HWTJ_JCREN_ENTRYNAMELEN_INV	<p>Meaning: Program error. The caller specified a value of zero as the length of the entry name buffer when the specified objectHandle was of type HWTJ_OBJECT_TYPE or specified a nonzero value when the specified objectHandle was of type HWTJ_ARRAY_TYPE.</p> <p>Action: Check for a probable coding error. Specify the actual length of the entry name buffer if the objectHandle is of type HWTJ_OBJECT_TYPE. Specify zero if the objectHandle is of type HWTJ_ARRAY_TYPE.</p> <p>Note: Specifying a bad entry name buffer length other than zero can result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
203 HWTJ_JCREN_ENTRYVALUEADDR_INV	<p>Meaning: Program error. The caller specified one of the following values:</p> <ul style="list-style-type: none"> • A value of zero for the address of the entry value buffer when the specified entryValueType was HWTJ_STRINGVALUETYPE, HWTJ_NUMVALUETYPE, HWTJ_JSONTEXTVALUETYPE • A nonzero value for the address of the entry value buffer when the specified entryValueType was HWTJ_OBJECTVALUETYPE, HWTJ_ARRAYVALUETYPE, HWTJ_TRUEVALUETYPE, HWTJ_FALSEVALUETYPE, or HWTJ_NULLVALUETYPE <p>Action: Check for a probable coding error. Specify the correct value based on the selected entryValueType. The parser enforces a zero value for this parameter whenever a value is specified that is incompatible with the entry value type the caller is trying to create.</p> <p>Note: Specifying a bad entry value buffer address other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>

Table 74. Return codes for the HWTJCREN service (continued)

Hexadecimal return code Equate symbol	Meaning and action
204 HWTJ_JCREN_ENTRYVALUELEN_INV	<p>Meaning: Program error. The caller specified one of the following values:</p> <ul style="list-style-type: none"> • A value of zero for the length of the entry value buffer when the specified entryValueType was HWTJ_STRINGVALUETYPE, HWTJ_NUMVALUETYPE, HWTJ_JSONTEXTVALUETYPE • A nonzero value for the length of the entry value buffer when the specified entryValueType was HWTJ_OBJECTVALUETYPE, HWTJ_ARRAYVALUETYPE, HWTJ_TRUEVALUETYPE, HWTJ_FALSEVALUETYPE, or HWTJ_NULLVALUETYPE <p>Action: Check for a probable coding error. Specify the correct value based on the selected entryValueType.</p> <p>Note: Specifying a bad entry value buffer length other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
205 HWTJ_JCREN_ENTRYVALUETYPE_INV	<p>Meaning: Program error. The caller specified an invalid entryValueType parameter.</p> <p>Action: Check for a probable coding error. Change the entryValueType to one of the valid values.</p>
206 HWTJ_JCREN_ENTRYNAME_INV	<p>Meaning: Program error. The caller specified an entry name, which contains a syntax error. (This is a valid return code for all types of the objectHandle parameter except HWTJ_ARRAY_TYPE.)</p> <p>Action: Check for a probable coding error. Consult Introducing JSON (http://json.org) or other JSON specifications to determine the correct syntax of a string or number value in a name or value pair.</p> <p>Note: The z/OS JSON parser adds the beginning and ending double quotation marks to all entry names. Any quotation marks contained in the entry name is flagged with this error.</p>
207 HWTJ_JCREN_ENTRYVALUE_INV	<p>Meaning: Program error. The caller specified an entry value, which contains a syntax error. (This is a valid return code for entryValueType parameter values HWTJ_STRINGVALUETYPE and HWTJ_NUMVALUETYPE.)</p> <p>Action: Check for a probable coding error. Consult Introducing JSON (http://json.org) or other JSON specifications to determine the correct syntax of a string or number value in a name or value pair.</p> <p>Note: The z/OS JSON parser adds the beginning and ending double quotation marks to all entry values with a type of HWTJ_STRINGVALUETYPE. Any quotation marks contained in the entry value buffer is flagged with this error.</p>

Table 74. Return codes for the HWTJCREN service (continued)

Hexadecimal return code Equate symbol	Meaning and action
F01 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJESCT — Encode or decode escape sequences (REXX)

Call the HWTJESCT service to encode or decode escape sequences in a JSON text stream.

Description

This service is a simple utility program, which transforms non-conforming JSON text (not properly escaped) into conforming JSON text (encode). This service can also be used to non-escape conforming JSON text into text, which can be easily read or displayed (decode).

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

REXX parameters

```
address hwtjson "hwtjesct"
                "ReturnCode",
                "RequestType",
                "EnOrDeSource",
                "EnOrDeTarget",
                "DiagArea."
```

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Character string

Contains the return code from the service.

RequestType

Supplied parameter.

- **Type:** Character string

A REXX variable that contains the request type. The request type can be set by either the **HWTJ_ENCODE** or the **HWTJ_DECODE** string.

EnOrDeSource

Supplied parameter.

- **Type:** Character string

A REXX variable that identifies the string to encode or decode.

EnOrDeTarget

Returned parameter.

- **Type:** Character string

A REXX variable that the service sets to the encoded or decoded JSON text.

DiagArea.

Returned parameter.

- **Type:** Stem variable

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in “HWTJESCT — Encode or decode escape sequences (REXX)” on page 438.

Table 75. Return codes for the HWTJESCT service.

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.

HWTJGAEN — Get array entry

Call the HWTJGAEN service to obtain a handle for an array entry.

Description

The HWTJGAEN service returns the handle of a particular array entry (**entryValueHandle**) as specified by the index of that array entry (**arrayEntryIndex**). The returned handle is used on subsequent calls to service calls to inquire about the value of this particular array entry.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGAEN service

All information for the HWTJGAEN service applies for REXX requests.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTJGAEN(ReturnCode, ParserHandle, ObjectHandle, ArrayEntryIndex, EntryValueHandle, DiagArea);	address hwtjson “hwtjgaen”, “ReturnCode”, “ParserHandle”, “ObjectHandle”, “ArrayEntryIndex”, “EntryValueHandle”, “DiagArea.”

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string

- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

ObjectHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a particular JSON array object. The **objectHandle** must have a JSON type of HWTJ_ARRAY_TYPE.

ArrayEntryIndex

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies the index of the n^{th} entry of the array object specified by **objectHandle**.

Note: This is a zero-origin index, meaning that the first entry in the array has an index of zero; the n^{th} entry has an index of $(n - 1)$.

EntryValueHandle

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

The handle representing the particular entry value selected in the JSON array object.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that may contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0002yyyy' for one of the following reasons:

yyyy **Reason**

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 76 on page 442.

Table 76. Return codes for the HWTJGAEN service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	<p>Meaning: Successful completion.</p> <p>Action: None.</p>
101 HWTJ_PARSERHANDLE_INV	<p>Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service).</p> <p>Action: Check for a probable coding error.</p>
102 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one will be allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service call abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. Only pass an objectHandle parameter that was returned by a prior z/OS JSON parser service call.</p>

Table 76. Return codes for the HWTJGAEN service (continued)

Hexadecimal return code Equate symbol	Meaning and action
105 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified objectHandle does not represent an array object (JSON type of HWTJ_ARRAY_TYPE).</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an objectHandle value that represents an array handle.</p>
107 HWTJ_INDEX_OUT_OF_BOUNDS	<p>Meaning: Program error. The index value specified by the arrayEntryIndex parameter is greater than the number of entries in the array.</p> <p>Action: Check for a probable coding error.</p> <ul style="list-style-type: none"> • Issue the HWTJGNUE (Get number of entries) service to determine the upper bound for the number of array entries. • Remember that the number of entries returned is the actual count of entries, but accessing a particular entry uses zero-origin indexing. To iterate through all the elements, the application must start at the first entry (index 0) up to the number of entries, minus 1.
F01 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJGBOV — Get boolean value

Call the HWTJGBOV service to obtain the boolean value of an entry.

Description

The HWTJGBOV service returns the boolean value of the entry, which is associated with a specified entry handle (**entryValueHandle**).

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGBOV service

All information for the HWTJGBOV service applies for REXX requests.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTJGBOV(ReturnCode, ParserHandle, EntryValueHandle, BooleanValue DiagArea);	address hwtjson “hwtjgbov”, “ReturnCode”, “ParserHandle”, “EntryValueHandle”, “BooleanValue”, “DiagArea.”

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

EntryValueHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a handle representing a particular entry in a JSON object that has a JSON type of boolean.

BooleanValue

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 1 byte (non-REXX)

A value contains one of the following values: HWTJ_true or HWTJ_false. Constants are provided in the associated interface definition file (IDF).

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0003yyyy' for one of the following reasons:

yyyy Reason

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 77.

Table 77. Return codes for the HWTJGBOV service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 77. Return codes for the HWTJGBOV service (continued)

Hexadecimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. Only pass an objectHandle parameter that was returned by a prior z/OS JSON parser service call.</p>
105 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified entryValueHandle does not represent an entry value with a JSON data type of HWTJ_BOOLEAN_TYPE.</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an entryValueHandle value that represents a boolean entry value handle.</p>

Table 77. Return codes for the HWTJGBOV service (continued)

Hexadecimal return code Equate symbol	Meaning and action
F01 HWTJ_INTERRUPT_STATUS_INV	Meaning: Program error. The calling program is disabled. The system rejects the service request. Action: Check the calling program for a probable coding error.
F02 HWTJ_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTJ_UNSUPPORTED_RELEASE	Meaning: The operating system level does not support this service. The system rejects the service request. Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.
FFF HWTJ_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTJGJST — Get JSON type

Call the HWTJGJST service to obtain the JSON data type of an entry or object.

Description

The HWTJGJST service returns the type of JSON data which is associated with the specified entry object or entry handle (**objOrEntryValueHandle**). This value can then be used by subsequent parse methods to take specific action based on the type of the data.

Example: If the **objOrEntryValueHandle** represents an object (JSONType = HWTJ_OBJECT_TYPE), the HWTJGNUE (Get number of entries) parse method can be used to determine the number of entries in the object. If the **objOrEntryValueHandle** represents a string (JSONType = HWTJ_STRING_TYPE), the HWTJGVAL (Get value) parse method can be used to get the actual string text.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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.

Requirement	Details
Locks:	No locks held.
Control parameters:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGJST service

All information for the HWTJGJST service applies for REXX requests.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTJGJST(ReturnCode, ParserHandle, ObjOrEntryValueHandle, JSONType, DiagArea);	address hwtjson “hwtjgst”, “ReturnCode”, “ParserHandle”, “ObjOrEntryValueHandle”, “JSONType”, “DiagArea.”

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

ObjOrEntryValueHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a handle representing either a JSON object or a JSON entry value. A JSON entry is a particular JSON name or value pair.

JSONType

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

A value representing the type of data of the item represented by the **objOrEntryValueHandle** parameter. Constants for the type values are defined in the supplied interface definition files (IDFs). For REXX, consult one of the other high-level language IDFs for a list of possible values.

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0004yyyy' for one of the following reasons:

yyyy Reason

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 78.

Table 78. Return codes for the HWTJGJST service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 78. Return codes for the HWTJGJST service (continued)

Hexadecimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objOrEntryValueHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. Pass either of the following values in the objOrEntryValueHandle parameter:</p> <ul style="list-style-type: none"> • An object value handle or entry value handle that was returned by a prior z/OS JSON parser service call • A zero for the root object
F01 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 78. Return codes for the HWTJGJST service (continued)

Hexadecimal return code Equate symbol	Meaning and action
F02 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJGNUE — Get number of entries

Call the HWTJGNUE service to obtain the number of entries (name or value pairs) associated with an object.

Description

The HWTJGNUE service returns the number of entries (name or value pairs) associated with the object represented by the specified object handle (**objectHandle**). The returned value is typically used on subsequent calls as a loop control to traverse all of the object elements.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGNUE service

All information for the HWTJGNUE service applies for REXX requests.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTJGNUE(ReturnCode, ParserHandle, ObjectHandle, NumOfEntries, DiagArea);	address hwtjson “hwtjgnue”, “ReturnCode”, “ParserHandle”, “ObjectHandle”, “NumOfEntries”, “DiagArea.”

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

ObjectHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a handle representing a particular JSON object (object or array object). The **objectHandle** value can be 0 (zero) for the root object or an entry value handle (**entryValueHandle**) whose JSON type is HWTJ_OBJECT_TYPE or HWTJ_ARRAY_TYPE.

NumOfEntries

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

A value represents the number of entries found in the object or array represented by the **objectHandle** parameter.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string
- **Length:** 132 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0005yyyy' for one of the following reasons:

yyyy Reason

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 79.

Table 79. Return codes for the HWTJGNUE service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 79. Return codes for the HWTJGNUE service (continued)

Hexadecimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. Pass either of the following values in the objectHandle parameter:</p> <ul style="list-style-type: none"> • An object handle or entry value handle that was returned by a prior z/OS JSON parser service call • A zero for the root object

Table 79. Return codes for the HWTJGNUE service (continued)

Hexadecimal return code Equate symbol	Meaning and action
105 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified objectHandle does not represent an entry value with a JSON data type of either HWTJ_OBJECT_TYPE or HWTJ_ARRAY_TYPE.</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an objectHandle value that represents either an object handle or an array handle.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJGNUV — Get number value (non-REXX)

Call the HWTJGNUV service to obtain the binary representation of a JSON number.

Description

The HWTJGNUV service returns the binary representation of a JSON number. The service returns a number in integer or floating-point format, based on the actual character-based number value represented by the specified entry value handle (**entryValueHandle**). This service may only be called when the JSON type for the **entryValueHandle** is HWTJ_NUMBER_TYPE. Consider using HWTJGVAL if no data conversion of a number is required.

If the value contains a decimal point or the scientific notation indicator (e or E), the service converts the value to a floating-point number. Conversely, if the value contains neither a decimal point nor the scientific notation indicator, the service converts the value to an integer number.

Underflow conditions: Converting a floating-point number may result in a number that is too small to be arithmetically represented in the buffer length as

HWTJGNUV

specified by the **valueBufferLen** (precision) parameter. This is known as an underflow condition, and this service will return a value of zero in this case. If this is not an acceptable outcome for the application and the **valueBufferLen** value can be increased, the application should increase the **valueBufferLen** value and reissue the request.

As with any floating-point number handling, use caution when performing floating-point conversions on certain types of data, such as currency values, as unexpected results may occur.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters on the CALL statement in the order shown.

Parameters

```
CALL HWTJGNUV(  
    ReturnCode,  
    ParserHandle,  
    EntryValueHandle,  
    ValueBufferAddr,  
    ValueBufferLen,  
    ValueDescriptor,  
    DiagArea);
```

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- **Type:** Integer

- **Length:** 4 bytes

Contains the return code from the service.

ParserHandle

Supplied parameter

- **Type:** Character string
- **Length:** 12 bytes

Specifies the JSON parser instance to be used.

EntryValueHandle

Supplied parameter

- **Type:** Integer
- **Length:** 4 bytes

Specifies a handle representing an entry in a JSON object that has a JSON type of NUMBER.

ValueBufferAddr

Supplied parameter

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of an 8-byte buffer where this service will store the converted number. After the service completes, the buffer contains the numeric binary representation of the number represented by the specified **entryValueHandle** parameter. The **valueDescriptor** parameter will indicate the format of the binary data pointed to by this parameter. The value could be a positive or negative 8-byte integer or an 8-byte floating-point number in IEEE binary floating-point format.

ValueBufferLen

Supplied parameter

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length (precision) of the buffer as specified by the **valueBufferAddr** parameter. The length specified here will determine the precision of the number returned by the service. The valid values are:

- 4** The service will return either a 4-byte signed integer or a 4-byte IEEE floating-point number.
- 8** The service will return either an 8-byte signed integer or an 8-byte IEEE floating-point number.

ValueDescriptor

Returned parameter

- **Type:** Integer
- **Length:** 4 bytes

A constant that indicates the format of the number being returned. Possible values include integer and floating point.

DiagArea

Returned parameter

- **Type:** Character string
- **Length:** 132 bytes

A storage area provided by the caller that may contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'000Dyyyy' for one of the following reasons:

yyyy Reason

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 80.

Table 80. Return codes for the HWTJGNUV service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.
102 HWTJ_PARSERHANDLE_INUSE	Meaning: Program error. Two possible reasons can result in this return code: <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. Action: Check for a probable coding error. <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one will be allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.

Table 80. Return codes for the HWTJGNUV service (continued)

Hexadecimal return code Equate symbol	Meaning and action
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service call abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the entryValueHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. Only pass an entryValueHandle parameter that was returned by a prior z/OS JSON parser service call.</p>
105 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified entryValueHandle does not represent an entry value with a JSON data type of HWTJ_NUMBER_TYPE.</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an entryValueHandle value that represents a number entry value handle.</p>
801 HWTJ_JGNUV_VALBUFFADDR_INV	<p>Meaning: Program error. The caller specified a value of zero as the address of the valueBufferAddr parameter.</p> <p>Action: Check for a probable coding error. Specify the actual address of the buffer where the returned converted binary representation of the JSON number is to be written.</p> <p>Note: Specifying a bad buffer address other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
802 HWTJ_JGNUV_VALBUFFLEN_INV	<p>Meaning: Program error. The caller specified an invalid value for the length of the output buffer.</p> <p>Action: Check for a probable coding error. Validate that the specified valueBufferLen value is one of the allowed values according to the description of the valueBufferLen parameter.</p> <p>Note: Specifying a bad buffer length other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>

Table 80. Return codes for the HWTJGNUV service (continued)

Hexadecimal return code Equate symbol	Meaning and action
803 HWTJ_JGNUV_NUM_OUT_OF_RANGE	<p>Meaning: The number cannot be converted. The attempt to convert the number to a binary representation results in an overflow condition (positive or negative).</p> <p>Action: If a valueBufferLen of 4 was specified, consider increasing the valueBufferLen to 8 and retrying the service. If 8 was already specified, the number is too large for the HWTJGNUV service to convert. Consider issuing HWTJGVAL to obtain the character representation of the number.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJGOEN — Get object entry

Call the HWTJGOEN service to obtain the address of the name portion and the entry value handle for an object entry.

Description

The HWTJGOEN service returns two items which are associated with the specified object handle (**objectHandle**) and object entry index (**objectEntryIndex**):

1. The address of the name portion of this entry (within the previously supplied JSON text)
2. The **entryValueHandle** of this entry, which can be used on subsequent service calls to inquire about the value of this particular object entry

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGOEN service

All information for the HWTJGOEN service applies for REXX requests except:

- EntryName replaces EntryNameBufferAddr and EntryNameBufferLen
- ActualNameLenReturned is not used

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTJGOEN(ReturnCode, ParserHandle, ObjectHandle, ObjectEntryIndex, EntryNameBufferAddr, EntryNameBufferLen, EntryValueHandle, ActualNameLenReturned, DiagArea);	address hwtjson “hwtjgoen”, “ReturnCode”, “ParserHandle”, “ObjectHandle”, “ObjectEntryIndex”, “EntryName”, “EntryValueHandle”, “DiagArea.”

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)

- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

ObjectHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a handle which represents a particular JSON object. The **objectHandle** value can be 0 (zero) for the root object or an entry value handle (**entryValueHandle**) whose JSON type is HWTJ_OBJECT_TYPE.

ObjectEntryIndex

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies the index of the n^{th} entry of the object specified by **objectHandle**.

Note: This is a zero-origin index, meaning that the first entry in the object has an index of zero; the n^{th} entry has an index of $(n - 1)$.

EntryName (REXX)

Returned parameter.

- **Type:** Character string

Contains the returned EntryName of the JSON object entry.

EntryNameBufferAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of a buffer provided to hold the returned entry name of the JSON object entry.

EntryNameBufferLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the buffer provided as specified by the **entryNameBufferAddr** parameter.

EntryValueHandle

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

A handle representing the particular entry value selected in the JSON object.

ActualNameLenReturned (non-REXX)

Returned parameter.

- **Type:** Integer
- **Length:** 4 bytes

The actual length of the returned entry name. If the size of the provided buffer is not large enough to contain the entry name (returnCode =HWTJ_BUFFER_TOO_SMALL), this returned value can be used to reissue the service with the proper buffer size.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0006yyyy' for one of the following reasons:

yyyy Reason

- 0000 The parameters passed by the caller are not in the primary address space.
- 0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 81.

Table 81. Return codes for the HWTJGOEN service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 81. Return codes for the HWTJGOEN service (continued)

Hexadecimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. Pass either of the following values in the objectHandle parameter:</p> <ul style="list-style-type: none"> • An object handle or entry value handle that was returned by a prior z/OS JSON parser service call • A zero for the root object

Table 81. Return codes for the HWTJGOEN service (continued)

Hexadecimal return code Equate symbol	Meaning and action
105 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified objectHandle does not represent an object with a JSON data type of HWTJ_OBJECT_TYPE.</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an objectHandle value that represents an object (non-array object) handle.</p>
106 HWTJ_BUFFER_TOO_SMALL	<p>Meaning: Program error. The buffer provided for the entry name is not large enough to contain the entire name. The value of the entryNameBufferLen parameter is too small.</p> <p>Action: Check for a probable coding error. Examine the actualNameLenReturned parameter to determine the required size for the entry name. Then, increase the size of the buffer specified by the entryNameBufferAddr and entryNameBufferLen parameters and reissue the service call.</p>
107 HWTJ_INDEX_OUT_OF_BOUNDS	<p>Meaning: Program error. The index value specified by the objectEntryIndex parameter is greater than the number of entries in the object.</p> <p>Action: Check for a probable coding error.</p> <ul style="list-style-type: none"> • Issue the HWTJGNUE (Get number of entries) service to determine the upper bound for the number of object entries. • Remember that the number of entries returned is the actual count of entries, but accessing a particular entry uses zero-origin indexing. To iterate through all the elements, the application must start at the first entry (index 0) up to the number of entries minus 1.
F01 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJGVAL — Get value

Call the HWTJGVAL service to obtain the location of a string or number within the JSON text.

Description

The HWTJGVAL service returns the exact location in the JSON source text of a string or number associated with the specified entry value handle (**entryValueHandle**). This service might be called whenever the JSON type for the **entryValueHandle** is either a string or numeric type.

Note: If the JSON type for the **entryValueHandle** is numeric and the binary representation (rather than the character representation) of the number is preferred, consider using the HWTJGNUV (get numeric value) service.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJGVAL service

All information for the HWTJGVAL service applies for REXX requests except:

- Value replaces ValueLocationAddr and ValueLen

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTJGVAL(ReturnCode, ParserHandle, EntryValueHandle, ValueLocationAddr, ValueLen, DiagArea);	address hwtjson "hwtjgval", "ReturnCode", "ParserHandle", "EntryValueHandle", "Value", "DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

EntryValueHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a handle representing an entry in a JSON object that has a JSON type of STRING or NUMBER.

Value (REXX)

Returned parameter.

- **Type:** Character string

The name of a REXX variable that the service sets to the entry value represented by the EntryValueHandle.

ValueLocationAddr (non-REXX)

Returned parameter.

- **Type:** Pointer
- **Length:** 4 bytes

The address of the location in the source JSON text where the value represented by **entryValueHandle** resides.

ValueLen (non-REXX)

Returned parameter.

- **Type:** Integer
- **Length:** 4 bytes

HWTJGVAL

The length of the wanted value referenced by the **valueLocationAddr** parameter.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0007yyyy' for one of the following reasons:

yyyy Reason

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 82.

Table 82. Return codes for the HWTJGVAL service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 82. Return codes for the HWTJGVAL service (continued)

Hexadecimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the entryValueHandle parameter is not valid.</p> <p>Action: Check for a probable coding error. Only pass an entryValueHandle parameter that was returned by a prior z/OS JSON parser service call.</p>
105 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified entryValueHandle does not represent an entry value with a JSON data type of HWTJ_STRING_TYPE or HWTJ_NUMBER_TYPE.</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an entryValueHandle value that represents either a string or number entry value handle.</p>

Table 82. Return codes for the HWTJGVAL service (continued)

Hexadecimal return code Equate symbol	Meaning and action
F01 HWTJ_INTERRUPT_STATUS_INV	Meaning: Program error. The calling program is disabled. The system rejects the service request. Action: Check the calling program for a probable coding error.
F02 HWTJ_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTJ_UNSUPPORTED_RELEASE	Meaning: The operating system level does not support this service. The system rejects the service request. Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.
FFF HWTJ_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTJINIT — Initialize a parser instance

Call the HWTJINIT service to prepare to parse a JSON text stream.

Description

This service must be invoked before any other z/OS JSON parsing service contained in the toolkit. The service prepares the memory space required by the z/OS JSON parser in the callers address space. The memory allocation created by HWTJINIT is known as a *parser instance*. If you need your program to parse more than one JSON data stream simultaneously, you can create more than one parser instance by invoking HWTJINIT as many times as necessary (once for each data stream). This allows the parser to track multiple JSON streams at the same time. The caller distinguishes one JSON stream from another by a unique JSON parser handle returned from each HWTJINIT call.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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.

Requirement	Details
Control parameters:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJINIT service

All information for the HWTJINIT service applies for REXX requests except:

- MaxParserWorkAreaSize is not used

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTJINIT(ReturnCode, MaxParserWorkAreaSize, ParserHandle, DiagArea);	address hwtjson “hwtjinit”, “ReturnCode”, “ParserHandle”, “DiagArea.”

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

MaxParserWorkAreaSize (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the maximum amount of storage, in bytes, that the parser can consume during parser functions. This allows the application to optionally set an upper threshold for the amount of storage the parser can consume in addition to the storage explicitly allocated by the callers application.

- A value of zero (recommended) specifies no limit to the work area size; the parser can use as much storage as necessary to successfully parse or create JSON text.
- A nonzero value does not necessarily cause the parser to obtain the specified amount of storage when the JSON instance is initialized, but allows the

parser to consume up to the specified amount of memory in the users memory area. If the specified value is less than the minimum amount required for the parser to operate, a return code of HWTJ_WARNING is returned and the **diagArea** parameter contains amount of storage obtained, which also serves as the maximum area the parser is able to use. The **diagArea** parameter also contains the amount of overhead storage required by the parser, which can be useful when calculating an appropriate value to specify. Consider the following formula as a rough estimate to determine the value to specify:

$$\text{maxParserWorkAreaSize} = (\text{number of name/value pairs in the JSON text} \times 30) + (\text{total length, in bytes, of all "names", string values and number values added to the JSON text via the HWTJCREN service}) + \text{overhead storage}$$

ParserHandle

Returned parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

A value generated by the parser that represents a handle to be used on all subsequent JSON parser services for this parser instance. This instance contains all of the data structures and storage areas required for the JSON parsing services to run efficiently. The REXX variable is updated by this service.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that may contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0008yyyy' for one of the following reasons:

yyyy Reason

- 0000 The parameters passed by the caller are not in the primary address space.
- 0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 83.

Table 83. Return codes for the HWTJINIT service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.

Table 83. Return codes for the HWTJINIT service (continued)

Hexadecimal return code Equate symbol	Meaning and action
4 HWTJ_WARNING	<p>Meaning: Warning. The <code>maxParserWorkAreaSize</code> value specified by the caller is smaller than the minimum required by the z/OS JSON parser.</p> <p>Action: The system creates the minimum size workarea (larger than the user-specified value) and informs the caller of the actual size in the <code>reasonDesc</code> field of the <code>diagArea</code>. Modify future invocations of the HWTJINIT service to specify a larger <code>maxParserWorkAreaSize</code> value based on the recommendation provided in the <code>reasonDesc</code> field.</p>
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a 0C4 system ABEND. Check the <code>diagArea</code> for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>
10B HWTJ_CANNOT_OBTAIN_WORKAREA	<p>Meaning: System error. The Storage Obtain service could not obtain the work area storage as required by the z/OS JSON parser during the HWTJINIT service call.</p> <p>Action: Consult the <code>diagArea</code> for the Storage Obtain failure return code and additional information found in the <code>diagDesc</code> section. Check to see if there is sufficient memory available in order for the parser to obtain the necessary amount of work area. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>

Table 83. Return codes for the HWTJINIT service (continued)

Hexadecimal return code Equate symbol	Meaning and action
FFF HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJPARS — Parse a JSON string

Call this service to assign JSON text to a parser instance, validate its syntax, and create an internal representation of it in the parser's memory.

Description

The HWTJPARS service performs the following functions:

1. It assigns the JSON text specified by the caller to a previously created parser instance. All prior JSON text associated with this JSON parser instance, if any, is disassociated with the instance.
2. It checks the JSON text input to ensure that the text conforms to the JSON syntax standards. See *Introducing JSON* (<http://json.org>) for a description of what comprises valid JSON text. Any errors or deviations from the published syntax standards are flagged as a parse error, and the caller is notified of both the kind of syntax error and the location in the JSON text input buffer where the reported error exists.
3. It creates an internal representation of the JSON text in the parser instance memory, allowing subsequent parse functions to be executed quickly. (The parser creates this internal representation of the JSON text in a DOM-like tree structure.)

Note: Reissuing the HWTJPARS service using the same parser handle as a prior HWTJPARS invocation will cause the parser instance to be reinitialized. All data saved in the internal representation of the JSON text, which is used by the parser for easy traversal by the other methods, will must be regenerated. In addition, the new HWTJPARS invocation causes all previous handles returned by the parser for objects or entries to be invalidated.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJPARS service

All information for the HWTJPARS service applies for REXX requests except:

- JSONText replaces JSONTextAddr and JSONTextLen

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
<pre>CALL HWTJPARS(ReturnCode, ParserHandle, JSONTextAddr, JSONTextLen, DiagArea);</pre>	<pre>address hwtjson "hwtjpars", "ReturnCode", "ParserHandle", "JSONText", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance associated with the supplied JSON text indicated by the **JSONTextAddr** parameter.

JSONText (REXX)

Supplied parameter.

- **Type:** Character string

Specifies the REXX variable that contains the JSON string to be parsed.

JSONTextAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes

HWTJPARS

Specifies the address of the actual buffer storage location of the JSON text to be parsed and associated with a previously initialized parser instance.

Notes:

1. The parser maintains a binding to the buffer specified by this parameter even after the service call to the HWTJPARS service completes. This buffer must not be freed or reused until the caller has completed all parsing functions associated with this JSON text string.
2. The JSON text supplied in the buffer should be encoded in EBCDIC (codepage 1047).

JSONTextLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the JSON text buffer pointed to by the **JSONTextAddr** parameter.

Note: The parser does not validate whether the specified length matches the length of the actual JSON text. The parser continues to parse the input up to the specified length. If the JSON text is shorter than the specified length, the parser generates a parse error if any non-null characters are encountered after the end of the JSON text. If the JSON text is longer than the specified length, the parser generates the appropriate parser error.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'0009yyyy' for one of the following reasons:

yyyy Reason

- | | |
|------|---|
| 0000 | The parameters passed by the caller are not in the primary address space. |
| 0001 | The number of parameters passed by the caller is incorrect. |

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 84 on page 477.

Table 84. Return codes for the HWTJPARS service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	<p>Meaning: Successful completion.</p> <p>Action: None.</p>
101 HWTJ_PARSERHANDLE_INV	<p>Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service).</p> <p>Action: Check for a probable coding error.</p>
102 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter, which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>

Table 84. Return codes for the HWTJPARS service (continued)

Hexadecimal return code Equate symbol	Meaning and action
108 HWTJ_WORKAREA_TOO_SMALL	<p>Meaning: Program error. The amount of JSON text to parse requires a larger work area than the size specified by the maxParserWorkAreaSize parameter on the HWTJINIT service.</p> <p>Action: Check for a probable coding error. IBM recommends specifying a maxParserWorkAreaSize value of 0 (unlimited work area size available to the parser). If the application cannot specify this value, then modify the invocation of the HWTJINIT service to specify a larger maxParserWorkAreaSize value based on the recommendations given in the diagDesc section of the diagArea and in the description of the HWTJINIT maxParserWorkAreaSize parameter.</p>
109 HWTJ_PARSE_ERROR	<p>Meaning: Supplied JSON text error. The JSON text passed by the caller to the HWTJPARS service contains a syntax error.</p> <p>Note: This error might also be generated if the specified JSONTextLen value is greater than the actual length of the JSON text and there are non-null (white space) characters after the JSON text.</p> <p>Action: Check the diagArea for a complete explanation of the error. The reasonCode portion of the diagArea pinpoints the reason for the parse failure, while the diagDesc portion points to the exact location in the supplied JSON text where the parser detected a JSON syntax error.</p>
10B HWTJ_CANNOT_OBTAIN_WORKAREA	<p>Meaning: System error. The Storage Obtain service could not obtain the work area storage as required by the z/OS JSON parser during the HWTJPARS service call.</p> <p>Action: Consult the diagArea for the Storage Obtain failure return code and additional information found in the diagDesc section. Check to see if there is sufficient memory available in order for the parser to obtain the necessary amount of work area. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
401 HWTJ_IPARS_JSONTEXTADDR_INV	<p>Meaning: Program error. The caller specified a value of zero for the address of the JSON text buffer.</p> <p>Action: Check for a probable coding error. Specify the actual address of the JSON text buffer.</p> <p>Note: Specifying a bad JSON buffer address other than zero might result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>

Table 84. Return codes for the HWTJPARS service (continued)

Hexadecimal return code Equate symbol	Meaning and action
402 HWTJ_IPARS_JSONTEXTLEN_INV	<p>Meaning: Program error. The caller specified a value of zero for the length of the JSON text buffer.</p> <p>Action: Check for a probable coding error. Specify the actual length of the JSON text buffer.</p> <p>Note: Specifying a bad JSON buffer length other than zero might result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
403 HWTJ_IPARS_WORKAREA_ERROR	<p>Meaning: System error. The z/OS JSON parser will sometimes need to issue system services to increase or decrease the amount of work area storage in order to properly build an internal representation of the JSON text data. When an error occurs while obtaining or releasing this storage, the parser cannot proceed.</p> <p>Action: Consult the diagArea for the return code of the failing Storage Obtain or Release service and additional information in the diagDesc section. Check to see if there is sufficient memory available in order for the application to obtain the necessary amount of work area storage. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJSERI — Serialize (build) JSON text

Call the HWTJSERI service to build the JSON text associated with a parser instance.

Description

The HWTJSERI service builds the JSON text associated with the specified parser instance by combining the existing JSON text (if any) and any newly added objects or entries.

The serialize service can be invoked any time that the application needs to have the complete JSON text representation associated with the parser instance. For example, the application might need to generate the JSON text in the following cases:

- The application needs to send the modified JSON text back to the partner application in the client/server web application model.
- The application needs to store the JSON text into a local data store for later use.
- The application has already created JSON text (using the HWTJCREN service) and needs to perform a search (HWTJSRCH service) with a **searchType** of HWTJ_SEARCHTYPE_GLOBAL to search through all existing and new JSON text. (A search with a **searchType** of HWTJ_SEARCHTYPE_GLOBAL cannot search through entries created by the HWTJCREN service.) In this case, a serialize request would be issued and the JSON text output would be used as input to the parse service (HWTJPARS). The newly parsed JSON text could then be searched globally (albeit with new handles).

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJSERI service

All information for the HWTJSERI service applies for REXX requests except:

- NewJSONText replaces NewJSONTextBufferAddr and NewJSONTextBufferLen
- ActualJSONTextLen is not used

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTJSERI(ReturnCode, ParserHandle, NewJSONTextBufferAddr, NewJSONTextBufferLen, ActualJSONTextLen, DiagArea);	address hwtjson "hwtjseri", "ReturnCode", "ParserHandle", "NewJSONText", "DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

NewJSONText (REXX)

Returned parameter.

- **Type:** Character string

REXX variable that contains the JSON text returned which is associated with the specified `parserHandle`.

NewJSONTextBufferAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of a buffer where the JSON text associated with the specified `parserHandle` is to be written.

NewJSONTextBufferLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the output buffer pointed to by the `newJSONTextBufferAddr` parameter.

ActualJSONTextLen (non-REXX)

Returned parameter.

- **Type:** Integer
- **Length:** 4 bytes

HWTJSERI

The actual length of the returned JSON text. If the size of the buffer specified by the **newJSONTextBufferLen** parameter is not large enough to contain the entire JSON text, a return code of HWTJ_BUFFER_T00_SMALL is returned. The application can then reissue the HWTJSERI request after the proper buffer size has been obtained.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that may contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'000Ayyyy' for one of the following reasons:

yyyy Reason

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 85.

Table 85. Return codes for the HWTJSERI service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 85. Return codes for the HWTJSERI service (continued)

Hexadecimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>
106 HWTJ_BUFFER_TOO_SMALL	<p>Meaning: Program error. The buffer provided for the new JSON text is not large enough to contain the generated text. The value specified for the newJSONTextBufferLen parameter is too small.</p> <p>Action: Check for a probable coding error. Examine the value returned in the actualJSONTextLen parameter to determine the necessary size for the JSON text buffer. Increase the size of the JSON text buffer, specify the larger size on the newJSONTextBufferLen parameter, and reissue the HWTJSERI request.</p>

Table 85. Return codes for the HWTJSERI service (continued)

Hexadecimal return code Equate symbol	Meaning and action
10A HWTJ_ROOT_OBJECT_MISSING	<p>Meaning: Program error. There is no JSON text to serialize.</p> <p>Action: Check for a probable coding error. Invoke the parse service (HWTJPARS) or the create service (HWTJCREN) to associate JSON text with the specified parser instance before invoking the HWTJSERI service to regenerate the JSON text.</p>
501 HWTJ_JSERI_NEWJTXTBUFFADDR_INV	<p>Meaning: Program error. The caller specified a value of zero for the address of the newJSONTextBufferAddr parameter.</p> <p>Action: Check for a probable coding error. Specify the actual address of the buffer to which the JSON text output is to be written.</p> <p>Note: Specifying a bad address for the JSON text output buffer that is other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
502 HWTJ_JSERI_NEWJTXTBUFFLEN_INV	<p>Meaning: Program error. The caller specified a value of zero for the length of the new JSON text output buffer.</p> <p>Action: Check for a probable coding error. Specify the actual length of the JSON text output buffer.</p> <p>Note: Specifying a bad length for the JSON text output buffer that is other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJSRCH — Search

Call the HWTJSRCH service to search for a particular name string.

JSON text is organized by name or value pairs in the form "*string*":*value*, where "*string*" is a descriptor of the value, and where *value* can be a string in double quotation marks, a number, a boolean value (true, false), a null value, an object, or an array. These structures can be nested.

It is often useful to quickly search for a particular name string in the JSON text. If a string is found, the traversal methods may then be useful to traverse items contained within the found object entry or in subordinate objects.

Description

The HWTJSRCH service searches for a particular "*name*" string within the entire JSON text or within a specific object. The starting point for the search can be either from the beginning of the text or from a specified handle start point.

Notes:

1. JSON array entries are simply a sequence of comma-separated values of any data type. As array entries do not have a "*name*" string, they cannot be searched by the HWTJSRCH service; however, the name of the array object itself can be searched, and nested objects with "*name*" strings within the specified array.
2. A global search can be used for searching through JSON text read by the parse service (HWTJPARS). Any text added to the string via the create service (HWJTCREN) cannot be searched globally, but might be searched with an object-scoped search.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See "Syntax, linkage, and programming considerations" on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJSRCH service

All information for the HWTJSRCH service applies for REXX requests except:

- SearchString replaces SearchStringAddr and SearchStringLen

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTJSRCH(ReturnCode, ParserHandle, SearchType, SearchStringAddr, SearchStringLen, ObjectHandle, StartingHandle, SearchResultHandle, DiagArea);	address hwtjson "hwtjsrch", "ReturnCode", "ParserHandle", "SearchType", "SearchString", "ObjectHandle", "StartingHandle", "SearchResultHandle", "DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be used.

SearchType

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies the scope of the search. Valid values are:

HWTJ_SEARCHTYPE_GLOBAL

Search the JSON text, starting at (but not including) the entry represented by the **startingHandle** parameter, for the first "name" that exactly matches the SearchString for REXX or the string pointed to by the SearchStringAddr parameter for non-REXX. The search is object-ignorant, meaning that there is no scoping of the search to be within the object specified by the **startingHandle** parameter. The search ends when either the search string is found or the end of the JSON text is reached, whichever occurs first.

HWTJ_SEARCHTYPE_OBJECT

Search the JSON text, starting at (but not including) the entry represented by the **startingHandle** parameter, for the first "name" within the object specified by the **objectHandle** parameter that exactly matches the SearchString for REXX or the string pointed to by the SearchStringAddr

parameter for non-REXX. The search ends when either the search string is found or the end of the object is reached, whichever occurs first.

SearchString (REXX)

Supplied parameter.

- **Type:** Character string

Specifies the search string name to be searched.

SearchStringAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes

Specifies the address of the location of the name string that the parser should attempt to locate within the JSON text.

Note: To search for an empty name string (""), specify a **searchStringAddr** value of zero. (In this case, the value of **searchStringLen** must also be zero.)

SearchStringLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the search string pointed to by the **searchStringAddr** parameter.

Note: To search for an empty name string (""), specify a **searchStringLen** value of zero. (In this case, the value of **searchStringAddr** must also be zero.)

ObjectHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies a handle representing a particular JSON object (object or array object) at which to start the search. An **objectHandle** is either a value of zero for the root object, or an **entryValueHandle** whose JSON type is HWJT_OBJECT_TYPE or HWJT_ARRAY_TYPE. This parameter must be specified for a **searchType** of HWJT_SEARCHTYPE_OBJECT and must be set to zero for a searchType of HWJT_SEARCHTYPE_GLOBAL.

StartingHandle

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies any handle value returned by a prior parser service call, or a value of zero.

If the **searchType** is HWJT_SEARCHTYPE_GLOBAL, the search starts at (but not including, unless the value is zero) the specified handle within the JSON text.

If the **searchType** is HWJT_SEARCHTYPE_OBJECT, the search starts at (but not including, unless the value is zero) the specified handle, if the **startingHandle** is either within the object specified by **objectHandle** or the **startingHandle** is zero (start searching at the beginning of the object).

If multiple instances of the same name string occur within the search scope, the **searchResultHandle** that is returned on one invocation of the search service can be used as the **startingHandle** for the next invocation.

SearchResultHandle

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

The returned handle that represents the next instance of the search string found after the specified starting point.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that may contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

If the string is not found, a “**not found**” return code is returned. If there are multiple strings with the same name string value, only the first is returned. The caller can issue another search request to find the next instance of this name string.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'000Byyyy' for one of the following reasons:

yyy Reason

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 86.

Table 86. Return codes for the HWTJSRCH service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The parserHandle parameter specified on the service call is not a valid parser handle (one that was returned by the HWTJINIT service). Action: Check for a probable coding error.

Table 86. Return codes for the HWTJSRCH service (continued)

Hexadecimal return code Equate symbol	Meaning and action
102 HWTJ_PARSERHANDLE_INUSE	<p>Meaning: Program error. Two possible reasons can result in this return code:</p> <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. <p>Action: Check for a probable coding error.</p> <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle might be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>
104 HWTJ_HANDLE_INV	<p>Meaning: Program error. The value specified for the objectHandle parameter is not valid, or a nonzero value objectHandle value was specified for a searchType of HWTJ_SEARCHTYPE_GLOBAL.</p> <p>Action: Check for a probable coding error.</p> <p>If the searchType is HWTJ_SEARCHTYPE_OBJECT, then pass either:</p> <ul style="list-style-type: none"> • An object handle or entry value handle on the objOrEntryHandle parameter that was returned on a prior z/OS JSON parser service call • A value of zero for the root object <p>If the searchType is HWTJ_SEARCHTYPE_GLOBAL, specify a value of zero for the objectHandle parameter.</p>

Table 86. Return codes for the HWTJSRCH service (continued)

Hexadecimal return code Equate symbol	Meaning and action
105 HWTJ_HANDLE_TYPE_ERROR	<p>Meaning: Program error. The specified objectHandle does not represent an object or array object (JSON data type of HWTJ_OBJECT_TYPE or HWTJ_ARRAY_TYPE).</p> <p>Action: Check for a probable coding error. Correct the mismatched handle and specify an objectHandle value that represents an object or array object handle.</p>
10A HWTJ_ROOT_OBJECT_MISSING	<p>Meaning: Program error. There is no JSON text to search.</p> <p>Action: Check for a probable coding error. Invoke the parse service (HWTJPARS) to associate JSON text with the specified parser instance before invoking the search service (HWTJSRCH).</p>
601 HWTJ_JSCH_SEARCHTYPE_INV	<p>Meaning: Program error. The caller specified an invalid searchType.</p> <p>Action: Check for a probable coding error. The caller should change the searchType value to one of the valid values.</p>
602 HWTJ_JSCH_SRCHSTRADDR_INV	<p>Meaning: Program error. The caller specified a value of zero for the address of the search string buffer.</p> <p>Action: Check for a probable coding error. Specify the actual address of the buffer containing the search string.</p> <p>Note: Specifying a bad search string buffer address other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>
603 HWTJ_JSCH_SRCHSTRLEN_INV	<p>Meaning: Program error. The caller specified a value of zero for the length of the search string buffer.</p> <p>Action: Check for a probable coding error. Specify the actual length of the search string buffer.</p> <p>Note: Specifying a bad search string buffer length other than zero may result in the parser terminating with a X'0C4' system ABEND. See the description of the HWTJ_INACCESSIBLE_PARM return code for more information.</p>

Table 86. Return codes for the HWTJSRCH service (continued)

Hexadecimal return code Equate symbol	Meaning and action
604 HWTJ_JSRCH_STARTINGHANDLE_INV	<p>Meaning: Program error. The specified startingHandle value is not valid for one of the following reasons:</p> <ul style="list-style-type: none"> • The caller specified a bad handle value that was not returned by a prior z/OS JSON parser service call. • If the searchType is HWTJ_SEARCHTYPE_OBJECT, the startingHandle value is either not zero or not the value of a valid handle within the object specified by the objectHandle parameter. <p>Action: Check for a probable coding error. Validate that the startingHandle parameter contains either a zero or a valid handle. Also, if the searchType is HWTJ_SEARCHTYPE_OBJECT, verify that the startingHandle is within the object specified by the objectHandle parameter.</p>
605 HWTJ_JSRCH_SRCHSTR_NOT_FOUND	<p>Meaning: The name string was not found in the search scope specified by the caller. If the searchType was HWTJ_SEARCHTYPE_GLOBAL, the name string was not found anywhere from the startingHandle to the end of the JSON text. If the searchType was HWTJ_SEARCHTYPE_OBJECT, the name string was not found anywhere in the object specified by objectHandle (from the startingHandle to the end of the object).</p> <p>Action: Check for a probable coding error. If the string was supposed to be found, verify that the searchType, startingHandle, and objectHandle (if applicable) are specified correctly. If all of these values are correct and the name string still cannot be found, verify that the JSON text being parsed actually contains the name string that the caller specified.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTJTERM — Terminate a parser instance

Call the HWTJTERM service to clean up resources obtained by a previous HWTJINIT invocation.

Description

When the services of a z/OS JSON parser instance are no longer needed, this service cleans up the resources in use by that parser instance. If this service is not invoked, the memory space allocated by the initialization and other parser services is allocated and ineligible for use by the application, and remains allocated until the job step task of the address space terminates.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task or SRB.
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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 421 for details about how to call the z/OS JSON parser services in the various supported programming languages.

REXX programming considerations for the HWTJTERM service

All information for the HWTJTERM service applies for REXX requests except:

- ForceOption is not used

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTJTERM(ReturnCode, ParserHandle, ForceOption, DiagArea);	address hwtjson “hwtjterm”, “ReturnCode”, “ParserHandle”, “DiagArea.”

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ParserHandle

Returned parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Specifies the JSON parser instance to be cleaned up. The **parserHandle** value was returned on a previous HWTJINIT service call for the particular JSON parser instance.

ForceOption (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Controls the behavior of the HWTJTERM service. Sometimes a parser instance can be stuck in an in-use state and cannot be terminated successfully. The in-use state can occur if a prior z/OS JSON parser service call resulted in an ABEND condition. This option allows the caller to force the parser instance to terminate. The valid values are:

HWTJ_NOFORCE

(Recommended) Terminates the parser instance and invalidates its parser handle only if the parser instance is not in an in-use state.

HWTJ_FORCE

Unconditionally terminates the parser instance and invalidates its parser handle, regardless of the in-use status of the parser instance.

Attention: Only use the HWTJ_FORCE option under the following conditions:

- No other threads in the address space are using this parser instance
- Multiple attempts to terminate the parser instance have resulted in a return code of HWTJ_PARSERHANDLE_INUSE.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'000Cyyyy' for one of the following reasons:

yyyy Reason

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 87.

Table 87. Return codes for the HWTJTERM service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTJ_OK	Meaning: Successful completion. Action: None.
101 HWTJ_PARSERHANDLE_INV	Meaning: Program error. The specified parserHandle parameter is not a valid parser handle (that is, one that was returned by the HWTJINIT service). Action: Check for a probable coding error.
102 HWTJ_PARSERHANDLE_INUSE	Meaning: Program error. Two possible reasons can result in this return code: <ol style="list-style-type: none"> 1. The parser handle is being used by another caller. Only one outstanding z/OS JSON parser service can use the same parser handle. 2. A previous service request using this parser handle resulted in an ABEND, and the parser instance was unable to indicate that its use of the parser handle has completed. Action: Check for a probable coding error. <ol style="list-style-type: none"> 1. While all z/OS JSON parser service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same parser handle, only one is allowed access. Change the application such that only one thread attempts to use the same parser handle at a time. 2. If the application detected an ABEND while the z/OS JSON parser was invoked, the parser instance associated with the parser handle could be permanently locked. To release the storage associated with the parser work area, issue the HWTJTERM service call with a forceOption of HWTJ_NOFORCE. If this fails with the same return code, issue another HWTJTERM service call with a forceOption of HWTJ_FORCE.

Table 87. Return codes for the HWTJTERM service (continued)

Hexadecimal return code Equate symbol	Meaning and action
103 HWTJ_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the parser. See “General programming considerations” on page 423 for details about z/OS JSON parser recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the parser abnormally ending with a X'0C4' system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the parser service calls abnormally ended. See “General programming considerations” on page 423 for details about actions to consider for this return code.</p>
701 HWTJ_JTERM_CANNOT_FREE_WORKA	<p>Meaning: System error. The Storage Release service could not release the work area storage as requested by the z/OS JSON parser.</p> <p>Action: Consult the diagArea for the Storage Release failure return code and additional information found in the diagDesc section. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
702 HWTJ_JTERM_FORCEOPTION_INV	<p>Meaning: Program error. The caller specified an invalid forceOption value.</p> <p>Action: Check for a probable coding error. The caller should change the forceOption value to one of the valid values, as described in “Parameters” on page 493.</p>
F01 HWTJ_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTJ_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTJ_UNSUPPORTED_RELEASE	<p>Meaning: The operating system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from this system, install it on a system that supports z/OS JSON parser services, and run the calling program again.</p>
FFF HWTJ_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Chapter 21. The z/OS HTTP/HTTPS protocol enabler

The HTTP/HTTPS protocol enabler portion of the z/OS client web enablement toolkit allows z/OS applications running in traditional environments to more easily participate as a web services client application.

Developers of new and existing z/OS applications can “webify” their applications using an industry standard API model of communicating with a web server. The enabler is a lightweight API that implements portions of the Hypertext Transfer Protocol 1.1 (HTTP/1.1), as specified by RFC 7230 (<http://tools.ietf.org/html/rfc7230>), RFC 7231 (<http://tools.ietf.org/html/rfc7231>), RFC 6265 (<http://tools.ietf.org/html/rfc6265>), and others. The toolkit also provides support for HTTP Secure (HTTPS), which layers HTTP over the Secure Sockets Layer (SSL) / Transport Layer Security (TLS) protocols to provide secure communications of standard HTTP requests over an open communications network.

HTTP/HTTPS enabler basics

The HTTP/HTTPS enabler portion of the toolkit encompasses two major aspects of a web services application:

- The *connection* to a server
- The *request* made to that server along with the response it returns

Connections

A toolkit connection is simply the network socket (pipeline) created between a client and server application for the purpose of exchanging information. A connection employs TCP sockets connecting an application to a remote IP address and port from an optionally-specified local IP address and port. This connection can either be established directly from a local IP address to the remote IP address, or can utilize a proxy server (an intermediary gateway that serves as a method of providing enhanced security and privacy controls). You can choose a standard socket connection or a secure connection that utilizes the SSL/TLS protocol. (See “Security considerations” on page 502.)

A connection must be established prior to any request being sent to the server. Once established, a connection may be dropped due a variety of factors. The toolkit will attempt to automatically reconnect to resume communications if it detects that the connection has been dropped.

An application may explicitly disconnect a connection. A successful disconnection causes all SSL and socket resources to be released.

Requests

In toolkit terms, a request is simply an HTTP request sent over a previously established connection. The toolkit provides some flexibility in the creation and usage of requests. Requests are not tightly coupled to a particular connection. Instead, a created request can be used with one or more connections. In addition, multiple requests can be created and kept in memory, each of which uses the same connection.

Requests can be specified with numerous options to customize the HTTP processing to the application’s preferences. A request method is selected along with the URI path name of the resource to be targeted by the HTTP

method. There are additional options to control the operation of the HTTP request. The following options are supported:

- Basic HTTP client authentication
- URI redirection (forwarding of HTTP requests from one URI to another)
- HTTP cookie management (a mechanism for the toolkit's cookie engine to maintain stateful information sent by the server on behalf of the client, which then can be sent back to the server on subsequent requests)
- Use of specific HTTP request headers

After a request is configured, it can be sent to the server by temporarily coupling itself with an existing connection on the send request call. Two configurable callback routines (user exits) can be set up to handle the response coming back from the server:

- A routine to receive control for each response header that is received
- A routine to receive control when a response body is present

The toolkit provides services that allow easy access to sockets, System SSL, and HTTP/HTTPS functionality. To accomplish this, the toolkit makes available to applications a set of services with a similar interface as the easy-to-use, open-source libcurl programming interface. Applications do not have to deal with most of the intricacies of socket, SSL, or HTTP programming. The toolkit takes care of many of these nuances, allowing application programs to have a higher level of abstraction.

Elements of the z/OS HTTP/HTTPS enabler

The z/OS HTTP enabler is organized into five types of services:

- **Initialize, reset, and terminate:** The purpose of these services is to prepare the memory space required by the z/OS HTTP enabler, to reset the space in preparation for reuse by a subsequent connection or request, or to free the space after these services are no longer needed. The memory allocation created by the initialize service is known as a *connection* or *request* instance, depending on the type of instance being created.
- **Set options:** This service prepares a connection or request instance with the desired configuration options. A user invokes this service multiple times, once for each option to be set. When all of the options have been set for the connection or request instance, the connect or send request service can be called to perform the respective operation, as specified by the various options that have been set.
- **Connect and disconnect:** These services are for connection instances and are used to establish or to disconnect a socket connection (pipeline) between the client and the server. If SSL/TLS use has been configured, these services will handle all SSL interactions to bring up or tear down the SSL connection. Once a connection has been created, it can then be used by the send request service to flow an HTTP request to the server connected.
- **Send request:** This service couples a request with a connection. A previously initialized and configured HTTP request is sent over a previously initialized and configured connection. The HTTP response from the server can be handled through the use of the HTTP response header and HTTP response body callback (exit) routines.
- **Set link list service:** This is a utility service that creates a linked list of data objects of the same type. This linked list data type is required by any option which expects a variable number of items to be associated with that attribute (as

described in the following example). A toolkit application will first create the link list (the HWTH_SLST_NEW function) to add the first item to the linked list. Additional items can be added (the HWTH_SLST_APPEND function) or the entire linked list can be deleted (the HWTH_SLST_FREE function).

Example: An application using the toolkit has the option to directly send particular HTTP headers (HWTH_OPT_HTTPHEADERS) to the server. Because it is reasonable that the application would want to specify more than one HTTP header to send, this option requires the headers to be in SLST format. The application will first create the linked list (SLST) specifying the first HTTP request header to be added to the request. If more request headers are needed, it will append those to the end of the previously created SLST. When all of the request headers are added, it will issue the HWTHSET API to set this option to the newly created SLST. The result is that the HWTH_OPT_HTTPHEADERS option is set to all of the headers specified by the linked list.

The general usage of the z/OS HTTP enabler services in an application follows this general order:

1. Create a connection instance, which returns a connection handle. (HWTHINIT)
2. Set the necessary connection options, such as URI of the server, SSL options, and so on, associated with this connection instance. (HWTHSET)
3. Connect to the server. (HWTHCONN)
4. Create a request instance, which returns a request handle. (HWTHINIT)
5. Set the necessary request options, such as the request type, server resource, HTTP response callback exits, and so on, associated with this request instance. (HWTHSET)
6. Send the previously defined request over the previously defined connection. If configured by the application, HTTP response routines are called during this service call to process the response data. (HWTHRQST)
7. Disconnect the previous connection. (HWTHDISC)
8. Free the work area associated with the request. (HWTHTERM)
9. Free the work area associated with the connection. (HWTHTERM)

Availability of the z/OS HTTP/HTTPS enabler

The z/OS HTTP enabler contained the z/OS client web enablement toolkit is available to any address space. The toolkit is enabled as part of z/OS initialization during IPL time. A message is written to the syslog regarding the status of the toolkit. Success or failure of toolkit initialization can be found by finding any HWT-prefixed syslog messages issued during IPL.

Syntax, linkage, and programming considerations

The z/OS HTTP enabler is available to many programs running in various address spaces. Many z/OS execution environments are supported, as well as various programming languages.

Programming interface files provided by the HTTP enabler

Table 88 on page 500 lists the programming interface files provided by the z/OS HTTP enabler.

Table 88. HTTP enabler

Programming language	Programming interface file
C / C++	Include file HWTHIC provided in SYS1.SIEAHDRV.H
COBOL	Copybook file HWTHICOB provided in SYS1.MACLIB
PL/I	Include file HWTHIPLI provided in SYS1.MACLIB
Assembler	Include file HWTHIASM provided in SYS1.MACLIB
REXX	See "HWTCONST — Initialize predefined variables (REXX)" on page 509 on how to access all the toolkit constants in REXX

Calling formats

Table 89 lists specific calling formats for languages that can invoke the z/OS HTTP enabler callable services.

Table 89. Calling formats for the z/OS HTTP enabler callable services

Programming language	Calling format
C / C++	<i>HTTPenabler_service_name (return_code,param1,param2,...)</i>
COBOL	CALL <i>HTTPenabler-service-name</i> USING <i>return_code,param1,param2,...</i>
PL/I	CALL <i>HTTPenabler_service_name (return_code,param1,param2,...)</i>
Assembler	CALL <i>HTTPenabler_service_name (return_code,param1,param2,...),VLIST</i>
REXX	ADDRESS HWTHHTTP " <i>HTTPenabler_service_name return_code parm1 parm2...</i> "

Linkage considerations

There are two ways for a compiled application to find the z/OS HTTP enabler callable services:

- **Linkage stub method**

(Recommended) Use the linkable stub routine HWTHCSS from SYS1.CSSLIB to link edits your object code. If you attempt to run the z/OS web enablement toolkit on a previous release of z/OS that does not support the z/OS HTTP enabler, this method results in the service call receiving a return code of X'F06' (HWTH_UNSUPPORTED_RELEASE).

- **Direct linkage method**

Code the linkage to the z/OS HTTP enabler services directly. This can be done if the program first confirms that the level of z/OS contains the toolkit. The following example shows the assembler linkage:

```
L   R14,CVTCVRT-CVT(R14,0)
L   R14,88(R14,0)
L   R15,4*HWT_SERV_XXXXX(R14,0)
LR  R14,R0
LR  R0,R13
BR  R15
```

In the example, *xxxxx* represents the last five letters of the service you want to call. This requires that the HWTHKASM assembler macro be included. If you attempt to run the HTTP enabler on a previous release of z/OS that does not support the HTTP enabler, this method results in the application receiving an abend X'019'.

Linkage considerations for high-level language programming

Callers must ensure that the proper linkage is made to the HTTP enabler services. The supplied IDF files for the various high-level languages contain the necessary definitions that ensure that the parameter list passed to the HTTP enabler has the high-order bit turned on for the last parameter. For example, for C, the linkage must be specified as OS linkage, such as:

```
#pragma linkage(HWTHxxx_CALLTYPE,OS)
```

For PL/I, the entry declaration should have the following options defined:

```
OPTIONS(LINKAGE(SYSTEM))
```

Linkage considerations for assembler language programming

Callers must also use the following linkage conventions:

- Register 1 must contain the address of a parameter list that is a list of consecutive words, each containing the address of a parameter to be passed. The last word in this list must have a 1 in the high-order (sign) bit.
- Register 13 must contain the address of an 18-word save area.
- Register 14 must contain the return address.
- Register 15 must contain the entry point address of the service being called.
- If the caller is running in AR ASC mode, access registers 1, 13, 14, and 15 must all be set to zero.
- On return from the service, general and access registers 2 - 14 are restored (registers 0, 1 and 15 are not restored).

Code page consideration

All input data into the HTTP enabler, except for body data, is assumed to be in EBCDIC encoding (code page 1047) before making any z/OS HTTP enabler service call. The toolkit translates any required data to meet any RFC code page standards when sending or receiving HTTP header data. Data specified in the request body or received in the response body is treated as-is unless the application requests the toolkit to translate from EBCDIC to ASCII or vice versa (see the documentation for HWTH_OPT_TRANSLATE_REQBODY and HWTH_OPT_TRANSLATE_RESPBODY later). Other than these options, the toolkit makes no attempt to translate this data into any format. It is the responsibility of the application to translate the data in either body into the correct encoding.

Environmental considerations

The following environmental considerations apply:

OMVS segment required: The toolkit uses TCP/IP sockets and Cryptographic Services System SSL services. Because both TCP/IP sockets and the SSL services require a z/OS UNIX (POSIX) environment, the HTTP enabler runs with a Language Environment (LE) POSIX(ON) environment. A POSIX(ON) environment requires the user ID associated with the address space using the HTTP enabler services to have an OMVS segment defined and associated with it. See the appropriate security product documentation, as applicable to your installation, for instructions on how to define an OMVS segment to a user. Failure to properly define the OMVS segment for the user invoking the HTTP enabler will likely result in an HWTH_ENVIRONMENTAL_ERROR return code.

z/OS UNIX limit of processes with a POSIX(ON) environment and its effect on concurrent connections: z/OS UNIX limits the number of concurrent POSIX(ON) environments that can be defined to a single address space. Since each HTTP enabler connection attempts to initialize a new POSIX(ON) environment (because of the environmental requirements listed earlier), each connection gets implicitly dubbed by LE. (*Dub* means to make a z/OS address space known to z/OS UNIX System Services. Once dubbed, an address space is considered to be a process.) If multiple connections are wanted from the same application (address space), a consideration of the dubbing configuration for the address space in which the HTTP enabler is running may be necessary.

The dubbing behavior that z/OS UNIX takes is customizable by using the z/OS UNIX set default service (BPX1SDD or BPX4SDD). If the dub default for the address space is set to DUBPROCESS, this allows concurrent subtasks to each have their own POSIX(ON) environment and, thereby, allow multiple connections from within the same address space, provided that each subtask has at most one connection.

Security considerations

There are several aspects of security to consider at both the connection level and the request level.

Connection security Considerations for connection security include:

TCP/IP stack and security product control

A toolkit application connecting to a server is limited by security profiles and definitions that are already in effect on the system where the application resides. Powerful controls, such as z/OS Communications Server NetAccess, in conjunction with security profiles defined using the SERVAUTH class, can be used to control network access authority, TCP/IP stack access authority, port access authority, and more. These security definitions can be configured to be as granular as required by an installation. See *z/OS Communications Server: IP Configuration Guide* and *z/OS Communications Server: IP Configuration Reference* for more information.

SSL/TLS connections

If requested by the application, the toolkit creates an SSL/TLS connection instead of a standard socket connection. These protocols provide data privacy and integrity, including server and client authentication, that is based on public key certificates. The toolkit uses z/OS System SSL services (part of the z/OS Cryptographic Services base component) to facilitate these SSL/TLS connections. (The term *SSL* is used throughout this publication to describe both the SSL and TLS protocols.) The certificate store configuration required by System SSL must be set up before making an SSL connection. The toolkit supports certificates stored in a key database file, a RACF key ring, or as a z/OS PKCS #11 token.

Request security: While an HTTP request flows over a secure SSL/TLS connection, the payload of the HTTP request is private between the client and the server. HTTP provides additional protocols to allow a user to authenticate with the server. In many cases, these additional authentication methods flows over an SSL connection exclusively to make sure that the security credentials are flowed from the client to the server in a private manner.

The toolkit implicitly supports *basic client authentication*, which packages a user ID and password in a format specified by RFC 1945 (<http://tools.ietf.org/html/>

rfc1945). The `HWTH_OPT_HTTPAUTH`, `HWTH_OPT_USERNAME`, and `HWTH_OPT_PASSWORD` settable options can be used to have the toolkit automatically create the necessary HTTP header.

Additional authentication schemes can be used explicitly by providing the specific HTTP headers and data. Authentication cookies can sometimes be used as a method by web servers to know whether the user is logged in or not, and the account with which they are logged in.

Problem determination considerations

Problem determination in a client/server web services application can be challenging. Here are a few debugging options that can aid in the debugging of your application:

Diagnostic area (diagArea): Each HTTP toolkit API requires the application to specify the `diagArea` parameter, a diagnostic output area that contains additional information that can be useful when a service fails with a nonzero return code or even in certain cases when the return code is zero. The `diagArea` includes three pieces of information: `HWTH_Service`, `HWTH_ReasonCode`, and `HWTH_ReasonDesc`.

HWTH_Service

A 4-byte constant value that indicates which internal service invoked by the toolkit detected the possible error. To determine which service, consult the first two bytes of this field and locate that value in the defined service constants provided in the IDFs (include files).

Note: The last two bytes are for IBM and can be provided to IBM Support, if necessary.

HWTH_ReasonCode

A specific error code returned by the internal service identified by `HWTH_Service`. This can be useful in determining the reason why a particular service failed.

HWTH_ReasonDesc

A 128-character field that contains a more detailed description of the problem. In many cases, this information is sufficient to determine the cause of the problem. In some cases, additional information is also provided here that you can provide to IBM Support if you cannot determine the problem after using the other problem determination techniques described here.

Verbose option: The HTTP enabler provides the option to get more information related to communications and data exchanged between the application and the web server by enabling the `HWTH_OPT_VERBOSE` option (setting this option to the `HWTH_VERBOSE_ON` value using the `HWTHSET` service). You can also select the destination of these trace messages. If you want to direct these messages to the application's standard output, no further action is necessary. If you want to direct the messages to a particular data set or file, you can use the `HWTHSET` service to set the `HWTH_OPT_VERBOSE_OUTPUT` option. (For more information about the `HWTH_OPT_VERBOSE_OUTPUT` option, see “HTTP/HTTPS enabler options and values” on page 552.) In many cases, these messages can help you quickly determine the root cause of a web application problem. Typically, you would use the verbose option during application debugging, then turn it off when the application is in production.

SOCKAPI interface: z/OS Communication Server provides a **SOCKAPI** CTRACE option, provided by TCP/IP, which can be used by application programmers to debug problems in their applications. The **SOCKAPI** option captures trace information that is related to the socket API calls that an application might issue. The **SOCKAPI** option is intended for use by TCP/IP support and provides information for debugging problems in the TCP/IP socket layer, z/OS UNIX System Services, or the TCP/IP stack. See *z/OS Communications Server: IP Diagnosis Guide* for more information about this problem determination methodology.

Recovery considerations

The z/OS HTTP enabler runs in the address space of the application. In addition, all the storage needed by the HTTP enabler is obtained in the application's address space. Because every application has its own programming environment, it is impossible for the HTTP enabler to predict the recovery environment required by the application.

The HTTP enabler provides a simple ESTAE recovery mechanism that provides a minimal recovery scheme for many environments from which the toolkit could be used. Programs that run under an FRR cannot avail themselves of the HTTP enabler recovery. Furthermore, the recovery provided by the HTTP enabler does not catch all errors. Therefore, it is imperative that a robust application provides its own recovery to catch any abnormal ends during toolkit execution.

When the HTTP enabler attempts to access application-provided parameters and those parameters are either inaccessible, point to an inaccessible location, or specify a length that goes beyond the available storage obtained by the application, an abend occurs. In many cases, the recovery of the toolkit catches the error and returns normally to the application with the **returnCode** parameter set to **HWTH_INACCESSIBLE_PARM** and the **reasonDesc** value in the **diagArea** set to the name of the bad parameter. There are some cases where parameter checking must occur outside of the HTTP enabler recovery environment. In these cases, the **returnCode** parameter is set to the error return code. If the toolkit abnormally ends, the application's recovery can consult the **returnCode** and **diagArea** values in the callers dynamic storage at the time of the abend and determine which parameter the toolkit could not process.

Note: Language Environment (LE) callers can see an LE-specific abend code other than 0C4. Under certain circumstances, the LE message, CEE3501S The module xxxxxxxx was not found, can appear in standard output, where xxxxxxxx is the application program's LE condition handler. The calling LE program still receives control with the failing toolkit return code and **diagArea** information.

Redirection considerations

The HTTP enabler supports automatic redirection (URL/URI forwarding) of requests from one location to another based on the HTTP specifications, as documented in the various RFC publications.

Cross-domain and non-cross-domain redirection: Redirecting a request can be as simple as reissuing an HTTP request to a resource on the same domain but with a different path (non-cross-domain redirection), or can be more involved by establishing a new connection to another domain and then reissuing the HTTP request to that new domain location (cross-domain redirection). The application can choose to allow or disallow cross-domain redirection by using the **HWTH_OPT_XDOMAIN_REDIRECTS** set option.

Redirection protocol change: A particular HTTP scheme (protocol) can sometimes be requested by a server when it notifies the client of the new redirect location. The application can choose what protocol changes are allowed by using the **HWTH_OPT_REDIRECT_PROTOCOLS** set option.

Number of redirect attempts: An application can limit the number of redirection that the toolkit is to attempt by setting the **HWTH_OPT_MAX_REDIRECTS** set option to a value. If no value is specified, the toolkit defaults to a maximum of five attempts.

General redirect behavior—some technical details: There is a number of valid redirect status response codes that can be sent from the server back to the client. The response to these status codes is dictated, usually, by the various RFCs that deal with HTTP redirect processing. The toolkit automatically processes redirects whenever it is safe to do so to unburden the application. The toolkit cannot automatically process an HTTP request that uses an unsafe method to protect the application. Table 90 describes how the toolkit handles each of the HTTP redirection status response codes (3xx), in accordance with current industry standards.

Table 90. Toolkit handling of HTTP redirection status response codes

Received HTTP status response codes	Toolkit behavior
<ul style="list-style-type: none"> • 300 Multiple Choices • 307 Temporary Redirect 	<p>The toolkit attempts to redirect the request to the location specified in the Location response header if and only if the application specifies a HWTH_OPT_REQUESTMETHOD value of HWTH_HTTP_REQUEST_GET, provided the other redirect options set by the application allow the toolkit to redirect the request.</p> <p>For other methods or if the Location header is not specified, the request is not redirected. In this case, the response header callback routine is driven for each response header along with the status response code. The application can choose to issue a subsequent request.</p>
<ul style="list-style-type: none"> • 301 Moved Permanently • 302 Found • 303 See Other 	<p>The toolkit automatically redirects the request, provided the other options set by the application allow the toolkit to redirect the request. If the application specifies a HWTH_OPT_REQUESTMETHOD value of HWTH_HTTP_REQUEST_GET, the redirect is identical to the initial request, except to the new target. However, if the HWTH_OPT_REQUESTMETHOD value is anything other than HWTH_HTTP_REQUEST_GET, the toolkit follows the industry de facto client behavior and downgrades the request to HWTH_HTTP_REQUEST_GET. Under no circumstances will an unsafe HWTH_OPT_REQUESTMETHOD value of HWTH_HTTP_REQUEST_POST, HWTH_HTTP_REQUEST_PUT, or HWTH_HTTP_REQUEST_DELETE be forwarded with the request method unchanged.</p>

REXX Programming Considerations

The toolkit provides a REXX host command environment, **HWTHHTTP**, to allow REXX applications to easily direct their requests to the HTTP enabler using an easy-to-use “made-for-REXX” interface. REXX applications running in TSO/E, System REXX, z/OS UNIX, or ISV-provided REXX environments are supported.

- To initialize the HWTHTTP host command environment in your REXX exec, it may be necessary to invoke the hwtcalls function at the beginning of your application: `call hwtcalls on`. After this invocation, both the ADDRESS HWTHTTP and ADDRESS HWTJSON host commands will direct API calls to the toolkit.
- To declare all toolkit constants in your REXX exec, use the HWTCONST service as documented below. Note: there is no REXX IDF (include file) provided by the toolkit.
- The toolkit services allocate task associated resources, which are released at task termination and the termination API calls.
- Handles are not shared among multiple tasks, which can restrict some reentrant REXX environments.
- HTTP handles can be updated by any of the HTTP services. The content of these variables should not be modified in any way by the application.
- Verify that all variables have proper content and are exposed if set outside of procedures.
- Variable names specified on toolkit REXX service calls are limited to 40 characters in length.
- REXX does not have unlimited variable content size. In general, a single variable cannot contain more than 16 MB of content. This limits the amount of data that can be sent and received in the HTTP request and response bodies. If the data required is greater than 16 MB for any of these cases, consider to use one of the high-level languages, which are supported by the toolkit (C/C++, COBOL, PL/I or Assembler).
- Programs running in any REXX environment that accesses z/OS UNIX from their address space codes CALL SYSCALLS 'SIGOFF' in their REXX exec before invoking any HTTP toolkit service (HWTH* services) to turn off MVS signaling. Failure to do this in your REXX exec can result in an HWTH_ENVIRONMENTAL_ERROR('F05'X) return code from the service.
- The built-in REXX RC variable contains the return code from the REXX HWTHTTP host command. This return code indicates the toolkits acceptance of the supplied REXX HWTHTTP host command. The return codes returned in the RC variable are generally unique to the REXX environment. In contrast, the HTTP service return code, the variable supplied on the service call itself, is only completed if the RC variable has a value of HWTH_OK (0) or HWT_REXXParmSyntaxError(1). Possible return codes returned by the toolkit in the RC variable are:

Host command return codes for the REXX environment:

Table 91. Host return codes for REXX

Host return code	Meaning and action
0	<p>Meaning: REXX toolkit host command successful.</p> <p>Action: Consult the toolkit return code on the service call to determine the final result of the request.</p>

Table 91. Host return codes for REXX (continued)

Host return code	Meaning and action
<p>1</p> <p>HWT_REXXParmSyntaxError</p>	<p>Meaning: REXX toolkit host command detects the parameter format is not in the proper form to be accepted.</p> <p>Action: Check for a probable coding error.</p> <ul style="list-style-type: none"> • See the return code on the toolkit service call to determine the reason for the syntax error. • See the REXX programming considerations of the toolkit service to see the exact calling specifications. • Compare the toolkit REXX service call attempted with service call examples in the supplied toolkit REXX programming sample found in SYS1.SAMPLIB. • See the DiagArea for further diagnostic information.
<p>2</p> <p>HWT_REXXUnsupportedService</p>	<p>Meaning: Program error. An unknown toolkit service name was specified on the toolkit REXX host command.</p> <p>Action: Check for a probable coding error. Specify a valid toolkit service name. For example, HWTHCONN.</p>
<p>3</p> <p>HWT_REXXInvalidNumOfParms</p>	<p>Meaning: Program error. The number of parameters specified on the toolkit REXX host command for the service name specified does not match the number of parameters expected.</p> <p>Action: Check for a probable coding error. See the REXX programming considerations of the toolkit service to see the exact calling specifications. Compare the toolkit REXX service call attempted with service call examples in the supplied toolkit REXX programming sample found in SYS1.SAMPLIB.</p>
<p>4</p> <p>HWT_REXXStemVarRequired</p>	<p>Meaning: Program error. The toolkit REXX service specified on the toolkit REXX host command is missing one or more required stem variables in the positional parameter list.</p> <p>Action: Check for a probable coding error. See the REXX programming considerations of the toolkit service to see the exact calling specifications. A stem variable parameter must specify a "." following the variable name. For example, "var.". Also, compare the toolkit REXX service call attempted with service call examples found in the supplied toolkit REXX programming sample found in SYS1.SAMPLIB.</p>
<p>5</p> <p>HWT_REXXParmNameTooLong</p>	<p>Meaning: Program error. One or more variables specified on the toolkit REXX service call on the toolkit REXX host command is greater than the toolkit maximum REXX variable length (40).</p> <p>Action: Check for a probable coding error. Reduce the variable name lengths on the toolkit REXX service call to be 40 characters or less in length</p>
<p>6</p> <p>HWT_REXXInvalidHostEnv</p>	<p>Meaning: System error. The toolkit detected an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Table 91. Host return codes for REXX (continued)

Host return code	Meaning and action
32 HWT_REXXUnexpectedError	<p>Meaning: System error. An unexpected error is detected. The system rejects the service call.</p> <p>Action: A symptom record has been written to LOGREC to record the problem. Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Using the REXX APIs in a z/OS UNIX Environment

Programs running in any REXX environment that accesses z/OS UNIX from their address space must turn off MVS signaling by coding CALL SYSCALLS "SIGOFF" in the REXX exec before invoking any HTTP toolkit service (HWTH* services). Failure to do this in your REXX exec can result in an HWTH_ENVIRONMENTAL_ERROR('F05'X) return code from the service.

z/OS HTTP enabler programming examples

The z/OS HTTP enabler provides a sample program in all supported programming languages to aid in the creation of applications that use the toolkit functions. Each sample contains examples of how to use almost all of the HTTP enabler services available in the toolkit. The samples are shipped in SYS1.SAMPLIB. Table 92 lists the sample files for each programming language.

Table 92. z/OS HTTP enabler programming sample files.

Programming language	Name of sample in SYS1.SAMPLIB
C / C++	HWTHXC1
COBOL	HWTHXCB1
PL / I	HWTHXPI1
REXX	HWTHXRX1

z/OS HTTP/HTTPS callable services

The z/OS HTTP enabler callable services are grouped under the following categories.

Initialization, reset, and termination services

Initialization, reset, and termination services deal with the creation and termination of HTTP enabler connection instances and request instances.

The HTTP enabler callable services in this category are:

- "HWTHINIT — Initialize an HTTP connection or request" on page 520
- "HWTHRSET — Reset an HTTP connection or request" on page 530
- "HWTHTERM — Terminate an HTTP connection or request" on page 547

Set options service

The set options service sets the options that are needed for a connection instance or request instance. The options are set one at a time. Thus, an application may call this service multiple times to set the connection options and to set the request options.

The HTTP enabler callable service in this category is:

- “HWTHSET — Set HTTP connection or request options” on page 535

Connect and disconnect services

The connect service attempts to connect to an HTTP server using all of the attributes associated with a connection handle, as previously set by the set options service. The disconnect service attempts to disconnect a connection previously created by the connect service.

The HTTP enabler callable services in this category are:

- “HWTHCONN — Connect to an HTTP server” on page 510
- “HWTHDISC — Disconnect from an HTTP server” on page 515

Send request service

The send request service sends an HTTP request to an HTTP server using a connection that was created by the connect service, and processes the response from the server.

The HTTP enabler callable service in this category is:

- “HWTHRQST — Send a request to an HTTP server” on page 524

Set link list service

The set link list service creates, appends, or frees a linked list, which is used to allow certain HTTP enabler option values to be represented by more than one data item.

The HTTP enabler callable service in this category is:

- “HWTHSLST — Linked list append service” on page 541

HWTCNST — Initialize predefined variables (REXX)

Call the HWTCNST service to initialize predefined variables in the current REXX variable pool.

Description

This service sets the variables with names prefixed for HWTH corresponding to the interface definition for the HTTP toolkit. This service is helpful when using symbolic names in checking for specific return codes or when specifying constant values in the application. The variable **HWT_CONSTANTS** is set to a list of the interface variable names, which is useful on a procedure expose statement to make the variables visible to a procedure.

HWTCNST

Note: This service also sets the variables for the z/OS JSON parser (HWTJ-prefixed) as well. If the REXX application utilizes both the HTTP and JSON parser portions of the toolkit, it is only necessary to call HWTCNST once to initialize all the variables.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters on the CALL statement in the order shown.

REXX parameters

```
address hwthttp "hwtcnst",  
               "ReturnCode",  
               "DiagArea."
```

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Character string

Contains the return code from the service.

DiagArea.

Returned parameter.

- **Type:** Stem variable

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in the "HWTCNST — Initialize predefined variables (REXX)" on page 428.

Table 93. Return codes for the HWTCNST service.

Hexadecimal return code Equate symbol	Meaning and action
0 HWTH_OK	Meaning: Successful completion. Action: None.

HWTHCONN — Connect to an HTTP server

Call the HWTHCONN service to connect to an HTTP server.

Description

The HWTHCONN service attempts to connect to an HTTP server using all of the attributes, which are associated with the supplied connection handle, as previously set by one or more calls to the HWTHSET (set options) service.

If the connection is successful, this connection is eligible to issue HTTP/HTTPS requests using the HWTHRQST service.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 499 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHCONN service

All information for the HWTHCONN service applies for REXX requests.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTHCONN(ReturnCode, ConnectionHandle, DiagArea);	address hwthttp "hwthconn", "ReturnCode", "ConnectionHandle", "DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter:

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ConnectionHandle

Supplied parameter:

HWTHCONN

- **Type:** Character string
- **Length:** 12 bytes

A value that was previously returned by an HWTHINIT call that specified a **handleType** of HWTH_HANDLETYPE_CONNECTION. The connection associated with this handle should have set the minimum number of connection options using the HWTHSET service before invoking the HWTHCONN service.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that can contain additional diagnostic information which is related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call can result in a X'04D' ABEND with a reason code of X'1001yyyy' for one of the following reasons:

yyyy Reason

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 94.

Table 94. Return codes for the HWTHCONN service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTH_OK	Meaning: Successful completion. Action: None.
4 HWTH_WARNING	Meaning: Possible error. The connect request was processed successfully, but detected a condition that should be reported back to the application. Action: Consult the DiagArea for a detailed explanation of this return code. Modify the application, as necessary.
101 HWTH_HANDLE_INV	Meaning: Program error. The value of the connectionHandle parameter that was specified on the service call is not a valid connect or request handle (one that was returned by the HWTHINIT service). Action: Check the calling program for a probable coding error.

Table 94. Return codes for the HWTHCONN service (continued)

Hexadecimal return code Equate symbol	Meaning and action
<p>102 HWTH_HANDLE_INUSE</p>	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> • The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. • A previous caller using this handle that is abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> • While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one is allowed access. Change the application so that only one thread attempts to use the same handle at the same time. • If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
<p>103 HWTH_HANDLETYPE_INVALID</p>	<p>Meaning: Program error. The application specified a request handle for the connectionHandle parameter.</p> <p>Action: Check the calling program for a probably coding error. Specify a valid connection handle for the connectionHandle parameter.</p>
<p>104 HWTH_INACCESSIBLE_PARM</p>	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service calls abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about actions to consider for this return code.</p>

Table 94. Return codes for the HWTHCONN service (continued)

Hexadecimal return code Equate symbol	Meaning and action
106 HWTH_COMMUNICATION_ERROR	<p>Meaning: A communication error has been detected. One or more of the following problems has occurred:</p> <ul style="list-style-type: none"> • A failure in the communication with the web server • An error in an underlying sockets or SSL/TLS service call • An error in obtaining the necessary system resources to process the connect. <p>Action: Check the diagArea for further diagnostic information. The toolkit uses many internal services, including sockets, SSL, and other calls when processing an HTTP API service call. If one of these internal services fails because of an error in communications with the targeted server or because of an internal environmental condition, the error is reported in the diagnostic area. This information can be useful to the application programmer but, in many cases, it is for the use of IBM Support. If one of these errors occurs, clean up the environment, check for possible communication configuration problems, and reissue the request. If the problem persists, contact the IBM Support Center.</p>
501 HWTH_HCONN_CONNECT_INV	<p>Meaning: Program error. One of the following errors occurred:</p> <ul style="list-style-type: none"> • The caller did not specify the required minimum parameters before connecting. • The caller specified incompatible or incomplete connection parameters. <p>Action: Check the calling program for a probable coding error. The diagArea should contain a detailed message explaining the problem.</p>
F01 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTH_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTH_MODE_INV	<p>Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWTH_AUTHLEVEL_INV	<p>Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services, which do not permit key 0 callers. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 94. Return codes for the HWTHCONN service (continued)

Hexadecimal return code Equate symbol	Meaning and action
F05 HWTH_ENVIRONMENTAL_ERROR	<p>Meaning: Language Environment (LE) could not create the proper environment for the request. This could occur for a number of reasons, the most likely of which are:</p> <ul style="list-style-type: none"> • The POSIX(ON) runtime option was not set (LE callers). • A POSIX(ON) environment was already established in the same address space, possibly because an HTTP connection was already established (non-LE callers). If the dubbing default is not set to DUBPROCESS, the limit is one POSIX(ON) environment per address space. If the dubbing default is set to DUBPROCESS, each thread in the address space can have its own POSIX(ON) environment, allowing for multiple connections. See “Environmental considerations” on page 501 for more information. <p>Action:</p> <ul style="list-style-type: none"> • For LE callers, verify that the POSIX(ON) runtime option has been enabled for the application. • For non-LE callers, verify the dubbing options selected for the address space and ensure that multiple POSIX(ON) runtime environments are not being requested. See “Environmental considerations” on page 501 for more information about how to enable this functionality.
F06 HWTH_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.</p>
FFF HWTH_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTHDISC — Disconnect from an HTTP server

Call the HWTHDISC service to disconnect from an HTTP server.

Description

The HWTHDISC service attempts to disconnect a connection created by the HWTHCONN service.

If the operation is successful, the connection is disconnected from the web server (all socket and SSL/TLS connections will be terminated), but all attributes associated with the connection handle remain intact. This allows subsequent HWTHCONN service calls to make minimal or no changes to the options for this connection, should a similar connection be desired in the future.

HWTHDISC

If the specified connection handle is not currently connected to a web server or has already been disconnected, the disconnect request ends with a successful (HWTH_OK) return code.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 499 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHDISC service

All information for the HWTHDISC service applies for REXX requests.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTHDISC(ReturnCode, ConnectionHandle, DiagArea);	address hwthttp “hwthdisc”, “ReturnCode”, “ConnectionHandle”, “DiagArea.”

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ConnectionHandle

Supplied parameter

- **Type:** Character string
- **Length:** 12 bytes

A value that was previously returned by an HWTHINIT call that specified a **handleType** of HWTH_HANDLETYPE_CONNECTION.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that may contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1002yyyy' for one of the following reasons:

yyyy Reason

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 95.

Table 95. Return codes for the HWTHDISC service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTH_OK	Meaning: Successful completion. Action: None.
4 HWTH_WARNING	Meaning: Possible error. The connect request was processed successfully, but detected a condition that should be reported back to the application. Action: Consult the DiagArea for a detailed explanation of this return code. Modify the application, as necessary.
101 HWTH_HANDLE_INV	Meaning: Program error. The value of the connectionHandle parameter that was specified on the service call is not a valid connect or request handle (one that was returned by the HWTHINIT service). Action: Check the calling program for a probable coding error.

Table 95. Return codes for the HWTHDISC service (continued)

Hexadecimal return code Equate symbol	Meaning and action
102 HWTH_HANDLE_INUSE	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> • The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. • A previous caller using this handle abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> • While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one will be allowed access. Change the application so that only one thread attempts to use the same handle at the same time. • If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
103 HWTH_HANDLETYPE_INVALID	<p>Meaning: Program error. The application specified a request handle for the connectionHandle parameter.</p> <p>Action: Check the calling program for a probably coding error. Specify a valid connection handle for the connectionHandle parameter.</p>
104 HWTH_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the toolkit. See the programming considerations in "Syntax, linkage, and programming considerations" on page 499 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service call abnormally ended. See the programming considerations in "Syntax, linkage, and programming considerations" on page 499 for details about actions to consider for this return code.</p>

Table 95. Return codes for the HWTHDISC service (continued)

Hexadecimal return code Equate symbol	Meaning and action
106 HWTH_COMMUNICATION_ERROR	<p>Meaning: A communication error has been detected. One or more of the following problems has occurred:</p> <ul style="list-style-type: none"> • A failure in the communication with the web server • An error in an underlying sockets or SSL/TLS service call • An error in obtaining the necessary system resources to process the disconnect <p>Action: Check the diagArea for further diagnostic information. The toolkit uses many internal services, including sockets, SSL, and other calls when processing an HTTP API service call. If one of these internal services fails because of an error in communications with the targeted server or because of an internal environmental condition, the error is reported in the diagnostic area. This information can be useful to the application programmer but, in many cases, it is for the use of IBM Support. If one of these errors occurs, clean up the environment, check for possible communication configuration problems, and reissue the request. If the problem persists, contact the IBM Support Center.</p>
109 HWTH_CONNECTION_NOT_ACTIVE	<p>Meaning: Program error. The HWTHDISC service cannot be issued for a connection that has not been made active by the HWTHCONN service.</p> <p>Action: Probable coding error. Only issue the HWTHDISC service call for connections that have been successfully connected using the HWTHCONN service.</p>
F01 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTH_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTH_MODE_INV	<p>Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWTH_AUTHLEVEL_INV	<p>Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services which do not permit key 0 callers. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 95. Return codes for the HWTHDISC service (continued)

Hexadecimal return code Equate symbol	Meaning and action
F05 HWTH_ENVIRONMENTAL_ERROR	<p>Meaning: Language Environment (LE) could not create the proper environment for the request. This could occur for a number of reasons, the most likely of which are:</p> <ul style="list-style-type: none"> • The POSIX(ON) runtime option was not set (LE callers). • A POSIX(ON) environment was already established in the same address space, possibly because an HTTP connection was already established (non-LE callers). If the dubbing default is not set to DUBPROCESS, the limit is one POSIX(ON) environment per address space. If the dubbing default is set to DUBPROCESS, each thread in the address space can have its own POSIX(ON) environment, allowing for multiple connections. See “Environmental considerations” on page 501 for more information. <p>Action:</p> <ul style="list-style-type: none"> • For LE callers, verify that the POSIX(ON) runtime option has been enabled for the application. • For non-LE callers, verify the dubbing options selected for the address space and ensure that multiple POSIX(ON) runtime environments are not being requested. See “Environmental considerations” on page 501 for more information about how to enable this functionality.
F06 HWTH_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.</p>
FFF HWTH_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTHINIT — Initialize an HTTP connection or request

Call the HWTHINIT service to initialize a connection with a web server or prepare to issue an HTTP request.

Description

The HWTHINIT service prepares to establish a connection with a remote web server or to prepare to issue an HTTP request. This service must be invoked before any other z/OS HTTP/HTTPS enabler service in the toolkit. The service prepares the memory space in the callers address space as required by either a z/OS HTTP enabler connection or a z/OS HTTP enabler request. Based on the specified **handleType**, the service returns either a connection handle or a request handle. If initializing a connection, a connection handle is passed back to the application, which can be used on subsequent services that reference this connection. Likewise,

if initializing a request, a request handle is passed back to the application, which can be used on subsequent services that reference this request.

Multiple connections and requests can be established for a single address space. See “z/OS UNIX limit of processes with a POSIX(ON) environment and its effect on concurrent connections” in “Environmental considerations” on page 501 for a discussion about the limitations on concurrent connections within a single address space.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 499 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHINIT service

All information for the HWTHINIT service applies for REXX requests.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTHINIT(ReturnCode, HandleType, ConnOrReqHandle, DiagArea);	address hwthttp "hwthinit", "ReturnCode", "HandleType", "ConnOrReqHandle", "DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode
 Returned parameter

HWTHINIT

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

HandleType

Supplied parameter

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies the type of handle to be initialized, from one of the following values:

HWTH_HANDLETYPE_CONNECTION

Initialize a connection to be used to connect to an HTTP server.

HWTH_HANDLETYPE_HTTPREQUEST

Initialize a request to be used to send HTTP requests to a web server.

ConnOrReqHandle

Returned parameter.

- **Type:** Character string
- **Length:** 12 bytes

Specifies a value generated by the toolkit representing a handle to be used on all subsequent HTTP enabler services for this connection or request instance. This instance contains all of the data structures and storage areas needed for the HTTP enabler services to run efficiently. The REXX variable is updated by this service.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area, which is provided by the caller might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1003yyyy' for one of the following reasons:

yyyy Reason

- 0000 The parameters passed by the caller are not in the primary address space.
- 0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 96 on page 523.

Table 96. Return codes for the HWTHINIT service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTH_OK	Meaning: Successful completion. Action: None.
103 HWTH_HANDLETYPE_INV	Meaning: Program error. The application specified an invalid value for the handleType parameter. Action: Check the calling program for a probable coding error. The caller should change the handleType to one of the valid values. See the IBM-supplied include files for the possible constant values that you can supply for this parameter.
104 HWTH_INACCESSIBLE_PARM	Meaning: Program error. The application passed an input or output parameter, which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about z/OS HTTP enabler recovery processing. Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service calls abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about actions to consider for this return code.
105 HWTH_CANNOT_OBTAIN_WORKAREA	Meaning: System error. The STORAGE OBTAIN service might not obtain the work area storage needed by the z/OS HTTP enabler during the HWTHINIT service call. Action: Consult the diagArea for the return code from the STORAGE OBTAIN service and additional information found in the diagDesc section. Ensure there is sufficient memory available for the toolkit to obtain the amount needed for the work area. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.
F01 HWTH_INTERRUPT_STATUS_INV	Meaning: Program error. The calling program is disabled. The system rejects the service request. Action: Check the calling program for a probable coding error.
F02 HWTH_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTH_MODE_INV	Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request. Action: Check the calling program for a probable coding error.

Table 96. Return codes for the HWTHINIT service (continued)

Hexadecimal return code Equate symbol	Meaning and action
F04 HWTH_AUTHLEVEL_INV	Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services, which do not permit key 0 callers. The system rejects the service request. Action: Check the calling program for a probable coding error.
F05 HWTH_ENVIRONMENTAL_ERROR	Meaning: Program error. The calling program is associated with a user ID that does not have an OMVS segment defined, or the proper z/OS UNIX environment is not available to this program. The system rejects the service request. Action: Ensure that the user ID under which this program is executing has an OMVS segment defined and that z/OS UNIX has been initialized.
F06 HWTH_UNSUPPORTED_RELEASE	Meaning: The system level does not support this service. The system rejects the service request. Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.
FFF HWTH_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTHRQST — Send a request to an HTTP server

Call the HWTHRQST service to send a request to an HTTP server.

Description

The HWTHRQST service sends an HTTP request represented by a request handle using a connection represented by the connection handle. When the server sends the response, the service processes the response and invoke the appropriate response callback (exit) routines, if specified. (See “Response callback (exit) routines (non-REXX)” on page 562 for more information.)

Upon completion of the service, the text in the returned **diagArea** parameter contains the HTTP status value of the request or other information about the result of the request.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = SASN.
AMODE:	31 bit.
ASC mode:	Primary or access register (AR).
Interrupt status:	Enabled for I/O and external interrupts.

Requirement	Details
Locks:	No locks held.
Control parameters:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 499 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHRQST service

All information for the HWTHRQST service applies for REXX requests except:

- The `StatusCode` and `ReasonCode` parameters are returned for REXX callers. These parameter descriptions are listed below for further explanation.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
<pre>CALL HWTHRQST (ReturnCode, ConnectionHandle, RequestHandle, DiagArea);</pre>	<pre>address hwthttp "hwthrqst", "ReturnCode", "ConnectionHandle", "RequestHandle", "StatusCode", "ReasonCode", "DiagArea."</pre>

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ConnectionHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes

A handle value that was previously returned by an HWTHINIT call that specified a **handleType** of `HWTH_HANDLETYPE_CONNECTION`.

RequestHandle

Supplied parameter.

- **Type:** Character string

HWTHRQST

- **Length:** 12 bytes

A handle value that was previously returned by an HWTHINIT call that specified a **handleType** of HWTH_HANDLETYPE_HTTPREQUEST.

StatusCode (REXX)

Returned parameter.

- **Type:** Character representation of an integer.

The name of a REXX variable that is set to the HTTP status code.

ReasonCode (REXX)

Returned parameter.

- **Type:** Character string

The name of a REXX variable that is set to the HTTP reason code.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area which is provided by the caller might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1004yyyy' for one of the following reasons:

yyyy Reason

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 97.

Table 97. Return codes for the HWTHRQST service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTH_OK	Meaning: Successful completion. Action: None.
4 HWTH_WARNING	Meaning: Possible error. The send request successfully received a response from the server, but detected a condition that should be reported back to the application. For instance, HWTHRQST returns this return code if the toolkit followed one or more redirects, or if the response header callback routine aborted further processing. Action: Consult the diagArea for a detailed explanation of this return code. Modify the application, as necessary.

Table 97. Return codes for the HWTHRQST service (continued)

Hexadecimal return code Equate symbol	Meaning and action
101 HWTH_HANDLE_INV	<p>Meaning: Program error. Either the connectionHandle or the requestHandle parameter that was specified on the service call is not a valid connect or request handle (one that was returned by the HWTHINIT service).</p> <p>Action: Check the calling program for a probable coding error.</p>
102 HWTH_HANDLE_INUSE	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> • The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. • A previous caller using this handle that is abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> • While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one is allowed access. Change the application so that only one thread attempts to use the same handle at the same time. • If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
103 HWTH_HANDLETYPE_INV	<p>Meaning: Program error. The application either specified a request handle for the connectionHandle parameter, or it specified a connection handle for the requestHandle parameter.</p> <p>Action: Check the calling program for a probable coding error. The diagArea indicates which handle parameter has the mismatch.</p>

Table 97. Return codes for the HWTHRQST service (continued)

Hexadecimal return code Equate symbol	Meaning and action
104 HWTH_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter, which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service calls abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about actions to consider for this return code.</p>
106 HWTH_COMMUNICATION_ERROR	<p>Meaning: A communication error has been detected. One or more of the following problems has occurred:</p> <ul style="list-style-type: none"> • A failure in the communication with the web server • An error in an underlying sockets or SSL/TLS service call • An error processing the HTTP request or the response coming back from the web server • An error in the translation of the data into the proper code page • An error in obtaining the necessary system resources to process the request. <p>Action: Check the diagArea for further diagnostic information. The toolkit uses many internal services, including sockets, SSL, and other calls when processing an HTTP API service call. If one of these internal services fails because of an error in communications with the targeted server or because of an internal environmental condition, the error is reported in the diagnostic area. This information can be useful to the application programmer but, in many cases, it is for the use of IBM Support. If one of these errors occurs, clean up the environment, check for possible communication configuration problems, and reissue the request. If the problem persists, contact the IBM Support Center.</p>
109 HWTH_CONNECTION_NOT_ACTIVE	<p>Meaning: Program error. The HWTHRQST service cannot be issued for a connection that has not been made active by the HWTHCONN service.</p> <p>Action: Probable coding error. Issue the HWTHCONN service to activate the connection specified on the HWTHRQST service call prior to the actual HWTHRQST call.</p>

Table 97. Return codes for the HWTHRQST service (continued)

Hexadecimal return code Equate symbol	Meaning and action
601 HWTH_HRQST_REQUEST_INV	<p>Meaning: Program error. One of the following errors occurred:</p> <ul style="list-style-type: none"> • The caller did not specify the required minimum parameters before issuing the send request. • The caller specified incompatible or incomplete request parameters. <p>Action: Check the calling program for a probable coding error. The diagArea should contain a detailed message explaining the problem.</p>
F01 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTH_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTH_MODE_INV	<p>Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWTH_AUTHLEVEL_INV	<p>Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services which do not permit key 0 callers. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 97. Return codes for the HWTHRQST service (continued)

Hexadecimal return code Equate symbol	Meaning and action
F05 HWTH_ENVIRONMENTAL_ERROR	<p>Meaning: Language Environment (LE) could not create the proper environment for the request. This could occur for a number of reasons, the most likely of which are:</p> <ul style="list-style-type: none"> • The POSIX(ON) runtime option was not set (LE callers). • A POSIX(ON) environment was already established in the same address space, possibly because an HTTP connection was already established (non-LE callers). If the dubbing default is not set to DUBPROCESS, the limit is one POSIX(ON) environment per address space. If the dubbing default is set to DUBPROCESS, each thread in the address space can have its own POSIX(ON) environment, allowing for multiple connections. See “Environmental considerations” on page 501 for more information. <p>Action:</p> <ul style="list-style-type: none"> • For LE callers, verify that the POSIX(ON) runtime option has been enabled for the application. • For non-LE callers, verify the dubbing options selected for the address space and ensure that multiple POSIX(ON) runtime environments are not being requested. See “Environmental considerations” on page 501 for more information about how to enable this functionality.
F06 HWTH_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.</p>
FFF HWTH_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HWTHRSET — Reset an HTTP connection or request

Call the HWTHRSET service to reset an HTTP connection or request.

Description

The HWTHRSET service returns a connection or a request back to the same state as when the HWTHINIT service was initially invoked.

- If you specify a connection handle, the connection is disconnected, if necessary, and all options that are previously set for this connection handle are undone. New HWTHRSET calls can now be made to set new connection options. No changes are made to any request handle.

- If you specify a request handle, all options which are previously set for this request handle are undone. New HWTHRSET calls can now be made to set new request options. No changes are made to any connection handle.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 499 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHRSET service

All information for the HWTHRSET service applies for REXX requests.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTHRSET (ReturnCode, ConnOrReqHandle, DiagArea);	address hwthttp "hwthrset", "ReturnCode", "ConnOrReqHandle", "DiagArea."

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

HWTHRSET

ConnOrReqHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes

Either a connection handle or a request handle that was previously returned by an HWTHINIT call. The REXX variable is updated by this service.

diagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1005yyyy' for one of the following reasons:

yyyy Reason

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 98.

Table 98. Return codes for the HWTHRSET service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTH_OK	Meaning: Successful completion. Action: None.
4 HWTH_WARNING	Meaning: Possible error. The connect request was processed successfully, but detected a condition that should be reported back to the application. Action: Consult the diagArea for a detailed explanation of this return code. Modify the application, as necessary.
101 HWTH_HANDLE_INV	Meaning: Program error. The value of the connOrReqHandle parameter that was specified on the service call is not a valid connect or request handle (one that was returned by the HWTHINIT service). Action: Check the calling program for a probable coding error.

Table 98. Return codes for the HWTHRSET service (continued)

Hexadecimal return code Equate symbol	Meaning and action
102 HWTH_HANDLE_INUSE	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> • The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. • A previous caller using this handle abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> • While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one will be allowed access. Change the application so that only one thread attempts to use the same handle at the same time. • If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
104 HWTH_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service call abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about actions to consider for this return code.</p>

Table 98. Return codes for the HWTHRSET service (continued)

Hexadecimal return code Equate symbol	Meaning and action
106 HWTH_COMMUNICATION_ERROR	<p>Meaning: A communication error has been detected. One or more of the following problems has occurred:</p> <ul style="list-style-type: none"> • A failure in the communication with the web server • An error in an underlying sockets or SSL/TLS service call • An error processing the HTTP request or response coming back from the web server • An error in the translation of the data into the proper codepage • An error in obtaining the necessary system resources to process the disconnect <p>Action: Check the diagArea for further diagnostic information. The toolkit uses many internal services, including sockets, SSL, and other calls when processing an HTTP API service call. If one of these internal services fails because of an error in communications with the targeted server or because of an internal environmental condition, the error is reported in the diagnostic area. This information can be useful to the application programmer but, in many cases, it is for the use of IBM Support. If one of these errors occurs, clean up the environment, check for possible communication configuration problems, and reissue the request. If the problem persists, contact the IBM Support Center.</p>
108 HWTH_CANNOT_FREE_WORKAREA	<p>Meaning: System error. The STORAGE RELEASE service could not release the work area storage or part of the work area storage, as requested by the z/OS HTTP enabler.</p> <p>Action: Consult the diagArea for the STORAGE RELEASE return code and for additional information found in the diagDesc section. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
F01 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTH_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTH_MODE_INV	<p>Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWTH_AUTHLEVEL_INV	<p>Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services which do not permit key 0 callers. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 98. Return codes for the HWTHRSET service (continued)

Hexadecimal return code Equate symbol	Meaning and action
F06 HWTH_UNSUPPORTED_RELEASE	Meaning: The system level does not support this service. The system rejects the service request. Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.
FFF HWTH_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTHSET — Set HTTP connection or request options

Call the HWTHSET service to set HTTP connection or request options.

Description

The HWTHSET service sets the necessary options required for a connection or a request. The options are set one at a time; a single HWTHSET service call sets one connection option or one request option. An application will likely call HWTHSET multiple times to set all the necessary connection options, and again for a request to set all the necessary request options.

Connection options for a connection handle must be set before calling the HWTHCONN service for this handle. Set service calls for a connection that occur after the connection has been established will generally have no effect until the connection handle has been disconnected and reconnected.

Note: The HWTH_OPT_COOKIE_TYPE HWTH_OPT_VERBOSE, HWTH_OPT_VERBOSE_OUTPUT, HWTH_OPT_COOKIE_INPUT_BUFFER and HWTH_OPT_COOKIE_OUTPUT_BUFFER options can be set at anytime.

Request options for a request handle must be completely set before calling the HWTHRQST service for this handle.

Note: It is possible to reset an individual option to its original state when the connection or request handle was first created by the HWTHINIT service by using the HWTHSET service. For non-REXX callers, if both optionValueAddr and optionValueLen are set to zero, the option is reset. For REXX callers, if the optionValue is set to null, the option is reset.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = SASN.
AMODE:	31 bit.

HWTHSET

Requirement	Details
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 and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 499 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHSET service

All information for the HWTHSET service applies for REXX requests except:

- OptionValue replaces OptionValueAddr and OptionValueLen.
- HWTH_OPT_RESPONSEHDR_EXIT and HWTH_OPT_RESPONSEBODY_EXIT options are not supported.
- HWTH_OPT_RESPONSEHDR_USERDATA and HWTH_OPT_RESPONSEBODY_USERDATA options have a different meaning than for the other languages supported by the toolkit. See these option descriptions in the section “HTTP/HTTPS enabler options and values” on page 552 for further information.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTHSET (ReturnCode, ConnOrReqHandle, Option, OptionValueAddr, OptionValueLen, DiagArea);	address hwthttp “hwthset”, “ReturnCode”, “ConnOrReqHandle”, “Option”, “OptionValue”, “DiagArea.”

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ConnOrReqHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes (non-REXX)

Either a connection handle or a request handle that was previously returned by a call to the HWTHINIT service.

Option

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies the option to be set. See “HTTP/HTTPS enabler options and values” on page 552 for a list of valid options and their descriptions. The values of the options are defined in the IBM-supplied files.

OptionValue (REXX)

Supplied parameter.

- **Type:** Character string.
- Specifies the REXX variable which contains the option value being set.
- See “HTTP/HTTPS enabler options and values” on page 552 for a list of valid options and their descriptions.

OptionValueAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes
- Specifies the address of the option value to be set. Generally, the value should be in the exact form that the server would expect to receive.

Note: If the option is an address itself or is an SLST, this parameter specifies the address of an address or the address of the SLST handle. Unless otherwise noted, the toolkit copies the contents of the buffer pointed to by this address into the toolkit’s work area. The application can modify the buffer contents after the service call has completed without adversely affecting the toolkit’s set value.

OptionValueLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the data pointed to by **optionValueAddr**. If the specified option is a constant value, the **optionValueLen** must be the length (in bytes) of the constant value, as specified in the IBM-supplied file. If the specified option is an address or an SLST, the **optionValueLen** must be 4.

DiagArea (non-REXX)**DiagArea. (REXX)**

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

HWTHSET

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1006yyyy' for one of the following reasons:

yyyy **Reason**

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 99.

Table 99. Return codes for the HWTHSET service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTH_OK	Meaning: Successful completion. Action: None.
101 HWTH_HANDLE_INV	Meaning: Program error. The value of the connOrReqHandle parameter that was specified on the service call is not a valid connect or request handle (one that was returned by the HWTHINIT service). Action: Check the calling program for a probable coding error.

Table 99. Return codes for the HWTHSET service (continued)

Hexadecimal return code Equate symbol	Meaning and action
<p>102 HWTH_HANDLE_INUSE</p>	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> • The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. • A previous caller using this handle abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> • While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one is allowed access. Change the application so that only one thread attempts to use the same handle at the same time. • If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
<p>104 HWTH_INACCESSIBLE_PARM</p>	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service call abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about actions to consider for this return code.</p>
<p>107 HWTH_CANNOT_INCREASE_WORKAREA</p>	<p>Meaning: System error. The STORAGE OBTAIN service could not obtain additional work area storage, as required by the z/OS HTTP enabler during the HWTHSET service call.</p> <p>Action: Consult the diagArea for the return code and additional information found in the diagDesc section. Ensure that there is sufficient memory available in order for the toolkit to obtain the necessary amount of work area storage. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

Table 99. Return codes for the HWTHSET service (continued)

Hexadecimal return code Equate symbol	Meaning and action
108 HWTH_CANNOT_FREE_WORKAREA	<p>Meaning: System error. The STORAGE RELEASE service could not release the work area storage or part of the work area storage, as requested by the z/OS HTTP enabler.</p> <p>Action: Consult the diagArea for the STORAGE RELEASE return code and for additional information found in the diagDesc section. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
201 HWTH_HSET_OPTIONVALADDR_INV	<p>Meaning: Program error. The specified optionValueAddr was zero, but the optionValueLen was non-zero.</p> <p>Action: Check the calling program for a probable coding error.</p>
202 HWTH_HSET_OPTIONVALLEN_INV	<p>Meaning: Program error. The specified optionValueLen was zero, but the optionValueAddr was non-zero.</p> <p>Action: Check the calling program for a probable coding error.</p>
203 HWTH_HSET_OPTION_INV	<p>Meaning: Program error. The caller specified an invalid option parameter, or the option parameter did not match the handle type associated with the connOrReqHandle parameter.</p> <p>Action: Check the calling program for a probable coding error. The caller should change the option value to one of the possible valid values and verify that the option is valid for the type of handle being specified. See the IBM-supplied include files for the valid constant values that can be supplied for this parameter.</p>
204 HWTH_HSET_OPTIONVALUE_INV	<p>Meaning: Program error. The caller specified an invalid option value (the value pointed to by optionValueAddr).</p> <p>Action: Check the calling program for a probable coding error. Each option is different. Some options have constants defined in the IBM-supplied include files that contain the range of possible values. Other options are character data that have certain rules which are enforced. Still others expect an address or an SLST. The diagArea will generally provide a specific reason why the toolkit did not accept the value that was specified. See "HTTP/HTTPS enabler options and values" on page 552 for more information.</p>
205 HWTH_HSET_CONN_ALREADY_ACTIVE	<p>Meaning: Program error. The specified connect option is not allowed when the connection associated with the connect handle has already been established. Most connection options may not be set after the HWTHCONN service has successfully completed.</p> <p>Action: Check the calling program for a probable coding error. Change the program to set this particular option prior to the HWTHCONN invocation for this connection.</p>
F01 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 99. Return codes for the HWTHSET service (continued)

Hexadecimal return code Equate symbol	Meaning and action
F02 HWTH_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTH_MODE_INV	Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request. Action: Check the calling program for a probable coding error.
F04 HWTH_AUTHLEVEL_INV	Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services which do not permit key 0 callers. The system rejects the service request. Action: Check the calling program for a probable coding error.
F06 HWTH_UNSUPPORTED_RELEASE	Meaning: The system level does not support this service. The system rejects the service request. Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.
FFF HWTH_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTHSLST — Linked list append service

Call the HWTHSLST service to create, append, or free a linked list of option values.

Description

The HWTHSLST service creates a linked list, appends to it, or frees an existing linked list. The linked list created by this service (called an SLST) is used to allow certain HTTP enabler toolkit option values to be represented by more than one data item. For example, the **HWTH_OPT_HTTPHEADERS** parameter takes an SLST as its input value, a set of one or more HTTP headers.

An application first creates an SLST, which creates the initial data structure and appends the first data item in a single service call. Subsequent calls to this service can append more data items to the existing SLST. The SLST can then be used as input to certain designated options. (See “HTTP/HTTPS enabler options and values” on page 552 to determine which options take an SLST as input). When the SLST is no longer needed, it can be freed.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = 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:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 499 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHSLST service

All information for the HWTHSLST service applies for REXX requests except:

- String replaces StringAddr and StringLen.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTHSLST (ReturnCode, RequestHandle, Function, sList, StringAddr, StringLen, DiagArea);	address hwthttp “hwthslst”, “ReturnCode”, “RequestHandle”, “Function”, “sList”, “String”, “DiagArea.”

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

RequestHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes

A handle value that was previously returned by a call to the HWTHINIT service that specified a **handleType** of HWTH_HANDLETYPE_HTTPREQUEST.

Function

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Specifies the requested operation. Valid values are:

HWTH_SLST_NEW

Create a new linked list, create the first element, and return an **sList** handle that can be used on subsequent HWTHSLST invocations.

Note: For the REXX variable, an **sList** handle is initialized and not returned.

HWTH_SLST_APPEND

Append a new linked list element to the specified **sList**.

HWTH_SLST_FREE

Delete a linked list.

sList

Supplied and returned parameter.

- **Type:** Integer (non-REXX), variable that identifies the sList (REXX).
- **Length:** 4 bytes (non-REXX)

Specifies the handle of the linked list that is to be created, modified, or deleted.

- When **function** is HWTH_SLST_NEW, this value must be zero upon input. The new **sList** handle is returned upon output.
- When **function** is HWTH_SLST_APPEND or HWTH_SLST_FREE, you must supply a valid **sList** handle on the service call.

StringAddr (non-REXX)

Supplied parameter.

- **Type:** Pointer
- **Length:** 4 bytes
- Specifies the address of the character string to be added to the end of the linked list chain.
 - When **function** is HWTH_SLST_NEW or HWTH_SLST_APPEND, specify a valid address.
 - When **function** is HWTH_SLST_FREE, specify zero.

StringLen (non-REXX)

Supplied parameter.

- **Type:** Integer
- **Length:** 4 bytes

Specifies the length of the data pointed to by **stringAddr**.

- When **function** is HWTH_SLST_NEW or HWTH_SLST_APPEND, specify a valid length.

- When **function** is HWTH_SLST_FREE, specify zero.

String (REXX)

Supplied Parameter.

- **Type:** Character string

Specifies the name of a REXX variable that contains the HTTP header to be added to the **sList**. This argument is required for each function but is not used on HWTH_SLST_FREE.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1007yyyy' for one of the following reasons:

yyyy **Reason**

- 0000 The parameters passed by the caller are not in the primary address space.
- 0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 100.

Table 100. Return codes for the HWTHSLST service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTH_OK	Meaning: Successful completion. Action: None.
101 HWTH_HANDLE_INV	Meaning: Program error. The value of the requestHandle parameter that was specified on the service call is not a valid request handle (one that was returned by the HWTHINIT service). Action: Check the calling program for a probable coding error.

Table 100. Return codes for the HWTHSLST service (continued)

Hexadecimal return code Equate symbol	Meaning and action
<p>102 HWTH_HANDLE_INUSE</p>	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> • The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. • A previous caller using this handle abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> • While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one is allowed access. Change the application so that only one thread attempts to use the same handle at the same time. • If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
<p>103 HWTH_HANDLETYPE_INV</p>	<p>Meaning: The application specified a connection handle for the requestHandle parameter.</p> <p>Action: Check for a probable coding error. Change the specified handle to be a request handle instead of a connection handle.</p>
<p>104 HWTH_INACCESSIBLE_PARM</p>	<p>Meaning: Program error. The application passed an input or output parameter, which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service calls abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about actions to consider for this return code.</p>

Table 100. Return codes for the HWTHSLST service (continued)

Hexadecimal return code Equate symbol	Meaning and action
107 HWTH_CANNOT_INCREASE_WORKAREA	<p>Meaning: System error. The STORAGE OBTAIN service could not obtain additional work area storage, as required by the z/OS HTTP enabler during the HWTHSLST service call.</p> <p>Action: Consult the diagArea for the return code and additional information found in the diagDesc section. Ensure that there is sufficient memory available in order for the toolkit to obtain the necessary amount of work area storage. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
108 HWTH_CANNOT_FREE_WORKAREA	<p>Meaning: System error. The STORAGE RELEASE service could not release the work area storage or part of the work area storage, as requested by the z/OS HTTP enabler.</p> <p>Action: Consult the diagArea for the STORAGE RELEASE return code and for additional information found in the diagDesc section. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
301 HWTH_SLST_SLIST_INV	<p>Meaning: Program error. The sList parameter specified on the service call is not a valid SLST. If the function is HWTH_SLST_NEW, the supplied sList was a nonzero value. If the function is either HWTH_SLST_APPEND or HWTH_SLST_FREE, the supplied sList is not a valid SLST handle that was returned from a previous HWTHSLST call with function = HWTH_SLST_NEW.</p> <p>Action: Check the calling program for a probable coding error.</p>
302 HWTH_HSLST_FUNCTION_INV	<p>Meaning: Program error. The specified function is not one of the valid function types.</p> <p>Action: Check the calling program for a probable coding error. The caller should change the function value to one of the possible valid values. See the IBM-supplied include files for the valid constant values that can be supplied for this parameter.</p>
303 HWTH_HSLST_STRINGLEN_INV	<p>Meaning: Program error. If function is either HWTH_SLST_NEW or HWTH_SLST_APPEND, the specified stringLen parameter was zero. If function is HWTH_SLST_FREE, the specified stringLen was nonzero.</p> <p>Action: Check the calling program for a probable coding error.</p>
304 HWTH_HSLST_STRINGADDR_INV	<p>Meaning: Program error. If function is either HWTH_SLST_NEW or HWTH_SLST_APPEND, the specified stringAddr parameter was zero. If function is HWTH_SLST_FREE, the specified stringAddr was nonzero.</p> <p>Action: Check the calling program for a probable coding error.</p>
F01 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 100. Return codes for the HWTHSLST service (continued)

Hexadecimal return code Equate symbol	Meaning and action
F02 HWTH_LOCKS_HELD	Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request. Action: Check the calling program for a probable coding error.
F03 HWTH_MODE_INV	Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request. Action: Check the calling program for a probable coding error.
F04 HWTH_AUTHLEVEL_INV	Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services which do not permit key 0 callers. The system rejects the service request. Action: Check the calling program for a probable coding error.
F06 HWTH_UNSUPPORTED_RELEASE	Meaning: The system level does not support this service. The system rejects the service request. Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.
FFF HWTH_UNEXPECTED_ERROR	Meaning: System error. The service encountered an unexpected error. The system rejects the service call. Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

HWTHTERM — Terminate an HTTP connection or request

Call the HWTHTERM service to terminate an HTTP connection or request.

Description

The HWTHTERM service cleans up the resources that were obtained by a previous call to the HWTHINIT service (including the entire work area) and invalidates its handle. If you do not invoke this service, the storage allocated by the HWTHINIT service and other HTTP enabler services remain allocated and ineligible to be used by the application, and remains allocated until the address space terminates.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key (except key 0).
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN = SASN.
AMODE:	31 bit.
ASC mode:	Primary or access register (AR).
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.

HWTHTERM

Requirement	Details
Control parameters:	Control parameters must be in the primary address space and addressable by the caller.
Linkage:	Standard MVS linkage conventions are used.

Programming requirements

See “Syntax, linkage, and programming considerations” on page 499 for details about how to call the z/OS HTTP/HTTPS enabler services in the various supported programming languages.

REXX programming considerations for the HWTHTERM service

All information for the HWTHTERM service applies for REXX requests.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters in the order shown.

Non-REXX parameters	REXX parameters
CALL HWTHTERM(ReturnCode, ConnOrReqHandle, ForceOption, DiagArea);	address hwthttp “hwthterm”, “ReturnCode”, “ConnOrReqHandle”, “ForceOption”, “DiagArea.”

Parameters

The parameters are explained as follows:

ReturnCode

Returned parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non-REXX)

Contains the return code from the service.

ConnOrReqHandle

Supplied parameter.

- **Type:** Character string
- **Length:** 12 bytes

Either a connection handle or a request handle that was previously returned by a call to the HWTHINIT service.

ForceOption

Supplied parameter.

- **Type:** Integer (non-REXX), character representation of an integer (REXX)
- **Length:** 4 bytes (non_REXX)

Controls the behavior of the HWTHTERM service. Sometimes a handle representing a connection or request can be stuck in an in-use state and cannot be terminated successfully. The in-use state can occur if a prior z/OS HTTP

enabler service call resulted in an ABEND condition. This option allows the caller to force the connection or request instance to terminate.

The valid values are:

HWTH_NOFORCE

(Recommended) Terminates the specified connection or request and invalidates its associated handle only if the connection or request is not currently in an in-use state.

HWTH_FORCE

Unconditionally terminates the specified connection or request and invalidates its associated handle, regardless of the in-use status of the connection or request.

Attention: Use the HWTH_FORCE option only under both the following conditions:

- No other threads in the address space are using this connection or request.
- Multiple attempts to terminate the connection or request have resulted in a return code of HWTH_HANDLE_INUSE.

DiagArea (non-REXX)

DiagArea. (REXX)

Returned parameter.

- **Type:** Character string (non-REXX), stem variable (REXX)
- **Length:** 136 bytes (non-REXX)

A storage area provided by the caller that might contain additional diagnostic information related to the service call. It consists of a 4-byte integer reason code field, a 4-byte integer service number field, and a 128-byte character string error text field.

ABEND codes

If the toolkit is unable to properly access the user-supplied parameter list, the call might result in a X'04D' ABEND with a reason code of X'1008yyyy' for one of the following reasons:

yyyy **Reason**

0000 The parameters passed by the caller are not in the primary address space.

0001 The number of parameters passed by the caller is incorrect.

Return codes

When the service returns control to the caller, GPR 15 and the **returnCode** parameter contain a hexadecimal return code, as listed in Table 101.

Table 101. Return codes for the HWTHTERM service

Hexadecimal return code Equate symbol	Meaning and action
0 HWTH_OK	Meaning: Successful completion. Action: None.

Table 101. Return codes for the HWTHTERM service (continued)

Hexadecimal return code Equate symbol	Meaning and action
4 HWTH_WARNING	<p>Meaning: Possible error. The connect request was processed successfully, but detected a condition that should be reported back to the application.</p> <p>Action: Consult the diagArea for a detailed explanation of this return code. Modify the application, as necessary.</p>
101 HWTH_HANDLE_INV	<p>Meaning: Program error. The value of the connOrReqHandle parameter that was specified on the service call is not a valid connect or request handle (one that was returned by the HWTHINIT service).</p> <p>Action: Check the calling program for a probable coding error.</p>
102 HWTH_HANDLE_INUSE	<p>Meaning: Program error. This return code results from one of the following reasons:</p> <ul style="list-style-type: none"> • The specified handle is being used by another caller. Only one outstanding z/OS HTTP enabler service can use the same handle. • A previous caller using this handle abnormally ended during an z/OS HTTP enabler service call and the toolkit was unable to indicate that its use of the supplied handle has completed. <p>Action: Check the calling program for a probable coding error.</p> <ul style="list-style-type: none"> • While all z/OS HTTP Enabler service calls are synchronous (blocking), if more than one task, process, or thread is running simultaneously and using the same handle, only one is allowed access. Change the application so that only one thread attempts to use the same handle at the same time. • If the application detected an abend while the z/OS HTTP enabler was invoked, the connection or request instance associated with the handle might be permanently locked. To release the storage associated with the handle work area, issue an HWTHTERM service call with a forceOption of HWTH_NOFORCE. If this fails with the same return code, issue another HWTHTERM service call with a forceOption of HWTH_FORCE.
104 HWTH_INACCESSIBLE_PARM	<p>Meaning: Program error. The application passed an input or output parameter which was inaccessible by the toolkit. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about z/OS HTTP enabler recovery processing.</p> <p>Action: Check for a probable coding error. Likely, the recovery of the caller detected this return code as a result of the toolkit abnormally ending with a 0C4 system ABEND. Check the diagArea for an explanation as to which parameter was attempting to be accessed when the toolkit service calls abnormally ended. See the programming considerations in “Syntax, linkage, and programming considerations” on page 499 for details about actions to consider for this return code.</p>

Table 101. Return codes for the HWTHTERM service (continued)

Hexadecimal return code Equate symbol	Meaning and action
108 HWTH_CANNOT_FREE_WORKAREA	<p>Meaning: System error. The STORAGE RELEASE service could not release the work area storage or part of the work area storage, as requested by the z/OS HTTP enabler.</p> <p>Action: Consult the diagArea for the STORAGE RELEASE return code and for additional information found in the diagDesc section. If the problem persists, search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>
401 HWTH_HTERM_FORCEOPTION_INV	<p>Meaning: Program error. The caller specified an invalid forceOption.</p> <p>Action: Check the calling program for a probable coding error. The caller should change the forceOption value to one of the possible valid values. See the IBM-supplied include files for the possible constant values that can be supplied for this parameter.</p>
F01 HWTH_INTERRUPT_STATUS_INV	<p>Meaning: Program error. The calling program is disabled. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F02 HWTH_LOCKS_HELD	<p>Meaning: Program error. The calling program is holding one or more locks. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F03 HWTH_MODE_INV	<p>Meaning: Program error. The calling program is running in a mode other than task, non-cross-memory mode. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>
F04 HWTH_AUTHLEVEL_INV	<p>Meaning: Program error. The calling program is running in key 0. The toolkit uses z/OS UNIX services, which do not permit key 0 callers. The system rejects the service request.</p> <p>Action: Check the calling program for a probable coding error.</p>

Table 101. Return codes for the HWTHTERM service (continued)

Hexadecimal return code Equate symbol	Meaning and action
F05 HWTH_ENVIRONMENTAL_ERROR	<p>Meaning: Language Environment (LE) could not create the proper environment for the request. This could occur for a number of reasons, the most likely of which are:</p> <ul style="list-style-type: none"> • The POSIX(ON) runtime option was not set (LE callers). • A POSIX(ON) environment was already established in the same address space, possibly because an HTTP connection was already established (non-LE callers). If the dubbing default is not set to DUBPROCESS, the limit is one POSIX(ON) environment per address space. If the dubbing default is set to DUBPROCESS, each thread in the address space can have its own POSIX(ON) environment, allowing for multiple connections. See “Environmental considerations” on page 501 for more information. <p>Action:</p> <ul style="list-style-type: none"> • For LE callers, verify that the POSIX(ON) runtime option has been enabled for the application. • For non-LE callers, verify the dubbing options selected for the address space and ensure that multiple POSIX(ON) runtime environments are not being requested. See “Environmental considerations” on page 501 for more information about how to enable this functionality.
F06 HWTH_UNSUPPORTED_RELEASE	<p>Meaning: The system level does not support this service. The system rejects the service request.</p> <p>Action: Remove the calling program from the system, and install it on a system that supports the z/OS HTTP enabler services. Then, run the program again.</p>
FFF HWTH_UNEXPECTED_ERROR	<p>Meaning: System error. The service encountered an unexpected error. The system rejects the service call.</p> <p>Action: Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.</p>

HTTP/HTTPS enabler options and values

The HTTP/HTTPS enabler portion of the z/OS web enablement toolkit allows many customizable options to determine how the toolkit should process a request.

Options can be set, one at a time, by using the HWTHSET service. (See “HWTHSET — Set HTTP connection or request options” on page 535.) The options are grouped by the type of handle (connection, request, or both) to which they apply.

Options for connections or requests

The following option allows the `connOrReqHandle` to have a `handleType` of `HWTH_HANDLETYPE_CONNECTION` or `HWTH_HANDLETYPE_HTTPREQUEST`:

HWTH_OPT_URI

The Uniform Resource Identifier (URI). This supplied buffer is the target location of the connection or the target resource of the request. This option is required for a connection, but is optional for a request. The option value depends on the **handleType** value, as follows:

When handleType is...	The valid values for HWTH_OPT_URI are...
HWTH_HANDLETYPE_CONNECTION	<p>Either an IPv4 or IPv6 address, or a hostname. Optionally, the hostname might be prefixed with the HTTP or HTTPS URI schemes (<code>http://</code> or <code>https://</code>). Use HWTH_OPT_PORT to specify this value. In technical terms, specify only the <i>authority</i> portion of the URI, minus the port name.</p> <p>Examples:</p> <pre>http://192.168.0.1 http://[2001:1890:1112:1::20] http://www.example.com</pre> <p>Note: The toolkit accepts single-byte EBCDIC character data for the value of the URI option. The use of multibyte character encodings (such as UTF-8 and UTF-16) is not supported. For more information, see “Code page consideration” on page 501.</p> <p>The toolkit indirectly supports Internationalized Domain Name (IDN). To specify a host name that contains Unicode characters, you must first apply the Punycode algorithm to convert the host name to an ASCII representation. (The Punycode algorithm converts each Unicode character outside of the US-ASCII character set to an encoded ASCII representation.) After the entire host name string has been encoded as an ASCII sequence, it is necessary to convert to EBCDIC before passing in the HWTH_OPT_URI set option, since the toolkit expects all input data to be in EBCDIC.</p>
HWTH_HANDLETYPE_HTTPREQUEST	<p>The name or resource (URN path portion) of the URI. The query and fragment portions of a URI may also be present.</p> <p>Examples:</p> <pre>/systems/z/ /over/here?name=abc#frag1</pre>

Options for connections only

The following options require a connection handle (**connectionHandle**).

General options

HWTH_OPT_VERBOSE

A 4-byte integer optionally used to turn on verbose messaging to aid in the understanding of application logic or debugging of network configuration problems. Valid values are:

HWTH_VERBOSE_OFF

The toolkit produces no additional trace messages. Analysis of

HTTP enabler options

application results rely on the **returnCode** and **diagArea** values from the toolkit or other tracing outside of the toolkit.

HWTH_VERBOSE_ON

The toolkit produces trace messages and directs them to the standard output for the application environment, unless directed elsewhere by the **HWTH_OPT_VERBOSE_OUTPUT** option.

Default: HWTH_VERBOSE_OFF

HWTH_OPT_VERBOSE_OUTPUT

An optional 1- 8 character name of a valid DD (data definition) statement that specifies where trace debugging messages are to be written. The toolkit only uses this option if the **HWTH_OPT_VERBOSE** option has been set to HWTH_VERBOSE_ON. The DD statement can specify one of the following destinations:

- A pre-allocated, traditional z/OS data set with the following attributes:
 - Physical sequential (DSORG=PS)
 - Unblocked, variable record format (RECFM=V), or undefined (RECFM=U)
 - Expandable (nonzero primary and secondary extents)
 - Disposition of OLD (DISP=OLD)
- A zFS or HFS file.

The toolkit automatically wraps the trace messages in the output data set or file when all available space has been consumed.

Default: None. (Debugging messages are directed to the application's standard output.)

Communication (socket) options

HWTH_OPT_PORT

An optional 4-byte integer indicating the remote port number to which to connect, instead of the default HTTP or HTTPS port.

HWTH_OPT_IPSTACK

An optional 1- 8 character local z/OS TCP/IP stack name to be used when communicating to the specified host name.

HWTH_OPT_LOCALIPADDR

An optional outgoing IP address from which the connection is to originate. This value should be in the same form as the **HWTH_OPT_URI** value for connections.

HWTH_OPT_LOCALPORT

An optional 4-byte integer indicating the outgoing port number from which the connection is to originate.

HWTH_OPT_SNDTIMEOUTVAL

An optional 4-byte integer to set a particular timeout value, in seconds, for the connection for outgoing requests.

Valid range: 1 - 2,678,400[®] seconds.

HWTH_OPT_RCVTIMEOUTVAL

An optional 4-byte integer to set a particular timeout value, in seconds, for the connection for incoming responses.

Valid range: 1 - 2,678,400 seconds.

Redirect options

HWTH_OPT_MAX_REDIRECTS

An optional 4-byte integer value that specifies the maximum number of redirects to follow. If zero, redirects are not allowed by the application.

Valid range: 0 - 50

Default: 5

HWTH_OPT_XDOMAIN_REDIRECTS

An optional 4-byte integer value that specifies the cross-domain redirect behavior. This option is only effective when **HWTH_OPT_MAX_REDIRECTS** has a value greater than zero. Valid values are:

HWTH_XDOMAIN_REDIRS_NOTALLOWED

The toolkit will attempt to follow a redirect if the redirect targets the current domain of the connection associated with the request.

HWTH_XDOMAIN_REDIRS_ALLOWED

The toolkit attempts to follow a redirect even if the domain of the redirect is different from the current domain of the connection associated with the request.

HWTH_OPT_REDIRECT_PROTOCOLS

An optional 4-byte integer value that specifies which additional protocols are allowed in the event that a redirect is received. Unlike many other options, calls to the set service that specify this option are cumulative. Multiple protocols can be specified by calling the set service multiple times, specifying one protocol at a time. This option is only effective when **HWTH_OPT_MAX_REDIRECTS** has a value greater than zero. Valid values are:

HWTH_REDIRECT_NOPROTCHANGE

Do not allow users to change protocols during a redirect. Effectively, this clears all prior set calls for this option and removes the default **HWTH_REDIRECT_HTTPS** as a valid redirect protocol.

HWTH_REDIRECT_HTTPS

Allow redirects to use the HTTPS protocol (SSL/TLS), even when the **HWTH_OPT_USE_SSL** value is set to **HWTH_SSL_NONE**. Valid SSL configuration information, such as **HWTH_OPT_SSLKEYTYPE** and other SSL support options must have been previously set in order for a redirect that requires SSL/TLS to succeed in the formation of an HTTPS connection.

HWTH_REDIRECT_HTTP

Allow redirects to use the HTTP protocol (non-SSL/TLS), even when the **HWTH_OPT_USE_SSL** value is set to **HWTH_SSL_USE**.

Note: Use this value carefully, as an SSL/TLS session can be downgraded to HTTP if this value is selected and a redirect to an HTTP host is requested.

HWTH_REDIRECT_NOHTTPS

Allow the user to override the default behavior of the toolkit and not allow redirects to use the HTTPS protocol.

HTTP enabler options

Default: HWTH_REDIRECT_HTTPS

SSL/TLS support options

HWTH_OPT_USE_SSL

An optional 4-byte integer value that determines whether SSL/TLS secure HTTPS communication is used. Valid values are:

HWTH_SSL_NONE

Do not attempt to use SSL/TLS communications (HTTP protocol only).

Note: SSL/TLS can still be used if redirects have been enabled (see the **HWTH_OPT_MAX_REDIRECTS** option for more information) and the **HWTH_OPT_REDIRECT_PROTOCOLS** option allows HTTPS. In this case, the SSL configuration specified by the SSL options available in the toolkit is used on a redirect to an HTTPS location.

HWTH_SSL_USE

Requires the connection to use SSL/TLS.

Note: SSL/TLS will always be used when connecting to the user-specified URI. A redirect URI could use the HTTP protocol only if redirects have been enabled (see the **HWTH_OPT_MAX_REDIRECTS** option for more information) and the **HWTH_OPT_REDIRECT_PROTOCOLS** option allows HTTP.

HWTH_OPT_SSLVERSION

An optional 4-byte integer value that sets one or more SSL versions to be supported by this HTTP request. Unlike many other options, calls to the set service that specify this option are cumulative. Multiple versions can be specified by calling the set service multiple times, setting one SSL version at a time. Valid values are:

HWTH_SSLVERSION_TLSv12

Support TLS version 1.2.

HWTH_SSLVERSION_TLSv11

Support TLS version 1.1.

HWTH_SSLVERSION_TLSv1

Support TLS version 1.0.

HWTH_SSLVERSION_SSLv3

Support SSL version 3.0. This is not recommended.

HWTH_SSLVERSION_DEFAULT

SSL is to be used but no security versions are sent on the request. The default SSL versions as determined by z/OS is sent. This option can also be used to reset all SSL version values after disconnecting, before reusing an existing connection handle.

Default: HWTH_SSLVERSION_DEFAULT

HWTH_OPT_SSLKEYTYPE

An optional 4-byte integer value that specifies the type of keystore to be used for HTTPS requests. This option is required when **HWTH_OPT_USE_SSL** is set to **HWTH_SSL_USE**. Valid values are:

HWTH_SSLKEYTYPE_KEYDBFILE

Use a key database file.

HWTH_SSLKEYTYPE_KEYRINGNAME

Use a SAF key ring name or a PKCS #11 token.

HWTH_OPT_SSLKEY

An optional buffer that specifies the name of the keystore to be used. This option is required when **HWTH_OPT_USE_SSL** is set to **HWTH_SSL_USE**. The value that you specify depends on the value of **HWTH_OPT_SSLKEYTYPE**, as follows:

When HWTH_OPT_SSLKEYTYPE is...	The valid value for HWTH_OPT_SSLKEY is...
HWTH_SSLKEYTYPE_KEYDBFILE	The path and file name of the key database file.
HWTH_SSLKEYTYPE_KEYRINGNAME	One of the following: <ul style="list-style-type: none"> A SAF key ring name, in the form <i>userid/keyring</i> A PKCS #11 token, in the form <i>*TOKEN*/token_name</i>

HWTH_OPT_SSLKEYSTASHFILE

Specifies the path and file name of the password stash file created by the System SSL gskkyman utility. This option is required when **HWTH_OPT_USE_SSL** is set to **HWTH_SSL_USE** and **HWTH_SSLKEYTYPE** is **HWTH_SSLKEYTYPE_KEYDBFILE**.

HWTH_OPT_SSLCLIENTAUTHLABEL

An optional label that represents a client certificate. If SSL client authentication is requested by the server, this option allows you to specify a client certificate other than the default client certificate to be used in the SSL handshake.

Proxy options

HWTH_OPT_PROXY

An optional buffer that specifies the HTTP proxy to use. This value should be in the same form as the **HWTH_OPT_URI** value for connections.

HWTH_OPT_PROXYPORT

An optional 4-byte integer indicating the proxy port to which to connect.

Cookie options

HWTH_OPT_COOKIEATYPE

An optional 4-byte integer value that specifies the cookie store engine behavior to be used. Unlike other options, this option takes effect immediately, even after the connection has been established. Valid values are:

HWTH_COOKIEATYPE_NONE

Turns off cookie support.

- Cookies set with the **HWTH_OPT_COOKIE** request option or cookies that are sent in the "Cookie" header as part of the **HWTH_OPT_HTTPHEADERS** option will be passed to the server, but will not be retained for future use.

HTTP enabler options

- For non-REXX, all cookies received in “Set-Cookie” response headers will be passed to the response header processor as specified by **HWTH_OPT_RESPONSEHDR_EXIT**, but will not be retained for future use by the toolkit.
- For REXX, all cookies received in “Set-Cookie” response headers will be passed to the application in the REXX stem specified by the **HWTH_OPT_RESPONSEHDR_USERDATA** option.
- If the toolkit cookie engine had been enabled previously, all cookies in the in-memory cookie store are deleted.

HWTH_COOKIE_TYPE_SESSION

Causes the toolkit cookie engine to activate and to start saving and sending session cookies for this connection. The cookies will be only available to the application while the connection is active and will not persist after the connection has ended.

- On each subsequent request, all eligible cookies that match the criteria specified by the cookie is propagated to the server. (An *eligible* cookie means a cookie that passes the expiration, domain, path, and secure filters).
- The **HWTH_OPT_COOKIE** request option sends any user-specified cookies in addition to any eligible cookies sent by the cookie engine, but they will not be retained by the engine.
- All cookies received in response headers will automatically be added as session cookies for this connection.
- If the application provides its own “Cookie” header as part of the **HWTH_OPT_HTTPHEADERS** option, this header is used and the cookie engine will not send any eligible cookies. In this case, all cookies specified by the request option are **HWTH_OPT_COOKIE** also ignored.
- The session cookies can be primed from an input cookie buffer by using the set services with the **HWTH_OPT_COOKIE_INPUT_BUFFER** option.

HWTH_COOKIE_TYPE_PERSIST

Causes the toolkit to start saving persistent cookies for this connection. If the **HWTH_OPT_COOKIE_OUTPUT_BUFFER** option has been set, the cookies are written to the application’s output cookie buffer during connection disconnect processing. If

HWTH_OPT_COOKIE_OUTPUT_BUFFER has not been set, this cookie type has the same behavior as **HWTH_COOKIE_TYPE_SESSION**.

- On each subsequent request, all eligible cookies that match the criteria specified by the cookie is propagated to the server. (An *eligible* cookie means a cookie that passes the expiration, domain, path, and secure filters).
- The **HWTH_OPT_COOKIE** request option sends any user-specified cookies in addition to any eligible cookies sent by the cookie engine, but they are not retained by the engine.
- All cookies received in response headers are automatically added as persistent cookies for this connection.
- If the application provides its own “Cookie” header as part of the **HWTH_OPT_HTTPHEADERS** option, this header is used and the cookie engine will not send any eligible cookies. In this case, all cookies specified by the request option are **HWTH_OPT_COOKIE** also ignored.

- The persistent cookies can be primed from an input cookie buffer by using the set services with the **HWTH_OPT_COOKIE_INPUT_BUFFER** option.

HWTH_OPT_COOKIE_INPUT_BUFFER

Optionally specifies a buffer containing a saved copy of the toolkit's cookie data store (cookie jar). This buffer can be used to prime the toolkit's cookie store for a new connection. (A previous connection with a **HWTH_OPT_COOKIEIETYPE** of **HWTH_COOKIEIETYPE_PERSIST** was disconnected, resulting in the toolkit writing its cookie store to a user-provided buffer, as specified by **HWTH_OPT_COOKIE_OUTPUT_BUFFER**).

- If you specify the same address for this option as for **HWTH_OPT_COOKIE_OUTPUT_BUFFER** and the **HWTH_OPT_COOKIEIETYPE** is set to **HWTH_COOKIEIETYPE_PERSIST**, the input buffer is overlaid.
- This parameter is ignored if **HWTH_OPT_COOKIEIETYPE** is set to **HWTH_COOKIEIETYPE_NONE**.
- If you are setting this option using the REXX API, the options specify the REXX variable containing prior output from a cookie output buffer.

HWTH_OPT_COOKIE_OUTPUT_BUFFER

Optionally specifies a buffer for cookies to be saved when a connection is disconnected and the **HWTH_OPT_COOKIEIETYPE** is set to **HWTH_COOKIEIETYPE_PERSIST**.

- If you specify the same address for this option as for **HWTH_OPT_COOKIE_INPUT_BUFFER** and the **HWTH_OPT_COOKIEIETYPE** is set to **HWTH_COOKIEIETYPE_PERSIST**, the input buffer is overlaid.
- This parameter is ignored if **HWTH_OPT_COOKIEIETYPE** is not set to **HWTH_COOKIEIETYPE_PERSIST**.
- If you are setting this option using the REXX API, the option specifies a REXX variable where the cookies are returned after a disconnect.

Guideline: The output buffer (cookie jar) specified here must be large enough to hold the cookies plus the metadata information about each cookie that is maintained by the cookie engine. Depending on how many cookies you plan to receive from the server (current maximum is 100), use the following formula to compute the size needed for the output buffer:

$$\text{number_of_cookies} \times (\text{average_cookie_data_size} + 1\text{K})$$

For instance, if the application is to hold 100 cookies with a maximum cookie size of 4 K bytes per cookie, the length of the output buffer would be: $100 \times (4 \text{ K} + 1 \text{ K}) = 500\text{K}$

Options for requests only

The following options require a request handle (**requestHandle**).

General request options

HWTH_OPT_HTTP_VERSION

An optional 4-byte integer specifying the wanted HTTP version. Valid values are:

HTTP enabler options

HWTH_HTTP_VERSION_NONE

The toolkit chooses the default value (currently, HTTP/1.1).

HWTH_HTTP_VERSION_1_0

Use HTTP/1.0.

HWTH_HTTP_VERSION_1_1

Use HTTP/1.1.

HWTH_OPT_REQUESTMETHOD

A required 4-byte integer specifying the wanted HTTP create, read, update, and delete (CRUD) request methods. Valid values are:

HWTH_HTTP_REQUEST_POST

Use the POST method.

HWTH_HTTP_REQUEST_GET

Use the GET method.

HWTH_HTTP_REQUEST_PUT

Use the PUT method.

HWTH_HTTP_REQUEST_DELETE

Use the DELETE method.

HWTH_OPT_HTTPHEADERS

An optional 4-byte sList handle, as returned by the HWTHSLST service, which contains a linked list of HTTP request headers. These headers are sent as-is, without any modification by the toolkit. If a header is specified that the toolkit might add by default (for instance, Host or Cookie), these headers take precedence.

Note: The HTTP headers specified in the sList must not be terminated by a carriage return, line feed (CRLF), as the toolkit automatically terminates each of the headers with a CRLF.

HWTH_OPT_REQUESTBODY

This is useful mainly on an HTTP PUT or POST operation.

Non-REXX:

An optional 4-byte pointer to a request body data buffer.

REXX:

A REXX variable name that contains the request body.

Note: Because the size of a request body could be substantial, the toolkit does not copy the buffer contents into the toolkit's work area. Therefore, any manipulation of the buffer data pointed to by this address after this option has been set and before the HWTHRQST service is called could yield undesired results.

HWTH_OPT_TRANSLATE_REQBODY

An optional 4-byte integer value that specifies the codepage translation behavior to be performed on a request body. Valid values are:

HWTH_XLATE_REQBODY_NONE

The toolkit will not translate the request body and send it as-is to the server.

HWTH_XLATE_REQBODY_E2A

The toolkit attempts to translate the application-provided request body from EBCDIC (code page 1047) to ASCII (code page ISO8859-1).

HTTP authorization options

HWTH_OPT_HTTPAUTH

An optional 4-byte integer value that specifies the HTTP authentication level to be used on this request. Valid values are:

HWTH_HTTPAUTH_NONE

No HTTP authorization is to be built by the toolkit.

HWTH_HTTPAUTH_BASIC

Use HTTP basic client authentication. The user and password, as specified by the **HWTH_OPT_USERNAME** and **HWTH_OPT_PASSWORD** options, is sent in the clear in the prescribed basic client authentication format.

Note: This option is only recommended for an HTTPS connection.

HWTH_OPT_USERNAME

An optional buffer which contains the user name to be used as part of various authentication protocols (such as basic client authentication), used with the **HWTH_OPT_HTTPAUTH** and **HWTH_OPT_PASSWORD** options. This option is required when **HWTH_OPT_HTTPAUTH** is set to **HWTH_HTTPAUTH_BASIC**.

HWTH_OPT_PASSWORD

An optional buffer which contains the password to be used as part of various authentication protocols (such as basic client authentication), used with the **HWTH_OPT_HTTPAUTH** and **HWTH_OPT_USERNAME** options. This option is required when **HWTH_OPT_HTTPAUTH** is set to **HWTH_HTTPAUTH_BASIC**.

Response options

HWTH_OPT_RESPONSEHDR_EXIT (non-REXX)

An optional 4-byte address of a program to receive control once for each response header received by the application. See “Response callback (exit) routines (non-REXX)” on page 562 for more information about how these callback (exit) routines operate.

HWTH_OPT_RESPONSEHDR_USERDATA

Non-REXX:

An optional buffer of user data to be passed into the response header exit when it receives control. This can serve as a communication mechanism for the application to specify that the response header exit is to behave in a certain manner when it receives control for the response that is associated with this request.

REXX:

An optional buffer of a REXX API variable where response headers are returned. If the variable is not a stem, the variable name is appended with the values similar to: *var0*, *var1*, *var1.1*,... If the variable is the name of the stem:

- stem.0 - contains the number of the headers returned
- stem.n where n is the number from 1-stem.0 - contains the header name
- stem.n.1 - contains the header value

HWTH_OPT_RESPONSEBODY_EXIT (non-REXX)

An optional 4-byte address of a program to receive control when the

HTTP enabler options

response body is received. See “Response callback (exit) routines (non-REXX)” for more information about how these callback (exit) routines operate.

HWTH_OPT_RESPONSEBODY_USERDATA

Non-REXX:

An optional buffer of user data to be passed into the response body exit when it receives control. This can serve as a communication mechanism for the application to specify that the response body exit is to behave in a certain manner when it receives control for the response associated with this request.

REXX:

The name of the REXX variable to contain the response body.

HWTH_OPT_TRANSLATE_RESPBODY

An optional 4-byte integer value that specifies the codepage translation behavior to perform on a response body. Valid values are:

HWTH_XLATE_RESPBODY_NONE

The toolkit will not translate response body received from the server.

HWTH_XLATE_RESPBODY_A2E

The toolkit attempts to translate the response body that is received from the server from ASCII (code page ISO8859-1) to EBCDIC (code page 1047).

Cookie options

HWTH_OPT_COOKIE

An optional buffer containing one or more cookies to be explicitly specified in an HTTP request. The format of the value string should be *name=contents*, where *name* is the cookie name and *contents* is the value of the cookie. You can specify more than one cookie by separating each with a semicolon (;).

This option can work in conjunction with the cookie engine enabled (that is, with **HWTH_OPT_COOKIE** set to **HWTH_COOKIE_SESSION** or **HWTH_COOKIE_PERSIST**). If enabled, any cookies specified by this option appear first in the list of cookies to be sent to the server, followed by any eligible cookies found by the cookie engine.

This option is ignored if the application has provided an explicit Cookie header as part of the **HWTH_OPT_HTTPHEADERS** option.

Response callback (exit) routines (non-REXX)

The toolkit provides a mechanism for an application to provide two user-defined callback (exit) routines that allow it to process the response, which comes back from the web server — a response header callback routine and a response body callback routine. Both of these callback routines are optional and are set by the application by using the **HWTHSET** service. The application has the option of passing user data to the callback routines. The toolkit copies the data into the work area when the user data is set and passes it into the callback routines as an input parameter at the time the response is received.

The callback routines must use standard z/OS linkage when receiving control from the toolkit (See “Linkage considerations for high-level language programming” on page 501 or “Linkage considerations for assembler language programming” on page 501

page 501 for the callback routine linkage requirements. Those instructions describe the same linkage requirements that these exits must follow). Failure to implement the proper entry and exit linkage in the callback routine could result in ABENDS from within the toolkit or cause the HWTHRQST service call to hang.

Use care to ensure that these exits do not take excessive time when processing the responses. Excessive processing time could cause the socket connection to time out or for the HWTHRQST service to hang for an undesired amount of time.

The exact parameter list specifications for these callback routines are described in the IBM-supplied, include files, listed in “Programming interface files provided by the HTTP enabler” on page 499.

Samples for both of these callback routines are provided in the sample code provided as specified in z/OS HTTP Enabler Programming Examples.

Response header callback routine

Whenever an application would like to know about all the response headers coming back from a given response. The application can set the **HWTH_OPT_RESPONSEHDR_EXIT** option to the address of a routine to process the headers. The callback routine is driven once for every response header, regardless of the overall status value sent back in the response. The status line of the response, the response header name, response header value, response header user data, and exit flags are passed as parameters to the callback routine. In addition, exit flags can be sent to the callback routine to indicate that a problem was detected during the processing of a particular response header before invoking the callback routine. The callback routine can react to these exit flags or ignore them. The callback routine also has the option of specifying a return code back to the toolkit to indicate whether to continue processing the response (**HWTH_RESP_EXIT_RC_OK**) or to abort further processing (**HWTH_RESP_EXIT_RC_ABORT**).

The response header callback routine (exit) normally gets driven one or more times (one for each response header) before the response body callback routine getting control (if data was sent from the server in the body). If the data returned from the server uses the chunk encoding transfer method and includes trailers, the header exit is driven once for each trailer found, with an additional indicator set in the exit flags to indicate that this particular header is, in fact, a trailer.

Response body callback routine

Whenever an application would like to know about the response body coming back from a given response. The application can set the **HWTH_OPT_RESPONSEBODY_EXIT** option to the address of a routine to process the body. The callback routine is driven once per request, regardless of the status value sent back in the response. The status line of the response, the response body, and response body user data are passed as parameters to the callback routine.

Under certain circumstances, the response body callback routine can be called with a response body having a length of zero. The programming of this callback routine should accommodate this possible condition.

The HTTP enabler supports the chunk encoding data transfer method (Transfer-encoding: chunk). The toolkit automatically de-chunks data that is sent from the server using the chunk encoding method. In other words, the response body exit does not need to handle the various chunks; rather, the data is delivered

Response callback (exit) routines

| to the exit already decoded. If the chunk data contains trailer headers, the header
| exit is invoked (once for each trailer header) before this routine receiving control.

| **Note:** The HTTP enabler ignores chunk extensions.
|

Part 10. SMF services

In addition to the callable services described in the following topics, you can find information about other SMF programming interfaces in *z/OS MVS System Management Facilities (SMF)*.

Chapter 22. SMF real-time interface

SMF provides an application programming interface (API) that offers real-time access to SMF in-memory resources.

The following callable services support real-time access to SMF records:

- “IFAMCON — Connect to an SMF in-memory resource”
- “IFAMGET — Obtain data from an SMF in-memory resource” on page 573
- “IFAMDSC — Disconnect from an SMF in-memory resource” on page 570
- “IFAMQRY — Query SMF in-memory resources” on page 577

The following steps describe the expected calling sequence:

1. Optional: Call the IFAMQRY service to determine which SMF in-memory resources are available to the application.
2. Call the IFAMCON service to connect to an in-memory resource.
3. Call the IFAMGET service in a loop to collect SMF records that are already in the in-memory resource or that are being recorded in real time.
4. Call the IFAMDSC service to disconnect from the in-memory resource and clean up resources.

For more information, see the chapter on using the SMF real-time interface in *z/OS MVS System Management Facilities (SMF)*.

IFAMCON — Connect to an SMF in-memory resource

Call the IFAMCON service to connect to an SMF in-memory resource.

Description

The IFAMCON service connects to SMF for in-memory data capture. The service establishes an environment to call the IFAMGET service to obtain SMF data from an in-memory resource. The caller must provide the target in-memory resource to access.

Notes:

- An active connection to an in-memory resource does not prevent the SET SMF=xx command from removing that resource from the configuration. No new data will be recorded to that resource; however, the resource is not removed from the configuration until the last connection disconnects. You can use the DISPLAY SMF,M command to display the connections to in-memory resources.
- An active connection prevents the SET SMF=xx command from changing the in-memory resource definition, such as changes to the **TYPE** or **RESSIZMAX** parameters.
- The system supports a maximum of 8 connections per in-memory resource, with a maximum of 128 connections per system.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.

Programming requirements

None.

Authorization

The caller requires READ access to the SAF resource protecting the in-memory resource to which the caller wants to connect.

Restrictions

None.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
Call IFAMCON (
                ConParmBlock,
                rc,
                rsn);
```

For assembler programming, the stub routine for this callable service resides in SYS1.CSSLIB.

Parameters

The parameters are explained as follows:

ConParmBlock

A connection parameter block.

Direction: Input/output

Type: Connection parameter block

The connection parameter block has the following format and is mapped by the IFAZSYSP macro in SYS1.MACLIB:

Decimal offset	Length	Type	Field	Description
0	4	EBCDIC	ID	ID with value of CNPB
4	2	binary	Length	Total length of the parameter block
6	1	binary	Unused	Unused; must be zero
7	1	binary	Version	Parameter block version number (X'01')
8	4	binary	Reserved	Reserved; must be zero
12	2	binary	Name length	Length of the name of the in-memory resource
14	26	EBCDIC	Name	Name of the in-memory resource, padded with blanks
40	16	binary	Reserved	Reserved; must be zero
56	16	binary	Token	Output token for access to the IFAMGET service
72	34	binary	Reserved	Reserved; must be zero

rc The return code from the service.

Direction: Output

Type: 4-word integer

rsn

The reason code from the service.

Direction: Output

Type: 4-word integer

ABEND codes

IFAMCON might abnormally end with system completion code X'353'. See *z/OS MVS System Codes* for an explanation and programmer response.

Related services

“IFAMDSC — Disconnect from an SMF in-memory resource” on page 570

Return and reason codes

When the service returns control to the caller, the **rc** parameter contains a hexadecimal return code and the **rsn** parameter contains a hexadecimal reason code, as listed in Table 102. The return code and reason code symbols are mapped by the IFARCINM macro.

Table 102. Return and reason codes for the IFAMCON service

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'00' IFAINMRetCodeOK	X'0000' IFAINMRsnCodeOK	Meaning: Successful completion. Action: None.

Table 102. Return and reason codes for the IFAMCON service (continued)

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'08' IFAINMRetCodeError	X'0801' IFAINMBadMode	Meaning: The caller is running in an incorrect mode for one or more of the following reasons: <ul style="list-style-type: none"> • The caller is not running in task mode. • The caller is in cross-memory mode. • The caller is holding a lock. Action: Change the program to run under a task in PASN = HASN mode with no locks held.
X'08' IFAINMRetCodeError	X'0802' IFAINMBadParmlist	Meaning: The parameter block is not accessible or has an incorrect format. Action: Correct the program to pass a valid parameter block.
X'08' IFAINMRetCodeError	X'0803' IFAINMNoConnections	Meaning: There are no available connections. Action: Determine the cause for the lack of connections.
X'08' IFAINMRetCodeError	X'0805' IFAINMUnSupported	Meaning: The caller is attempting to pass unsupported options in the parameter block. Action: Initialize all unused fields in the parameter block to zero.
X'08' IFAINMRetCodeError	X'0807' IFAINMNoSuchResource	Meaning: The specified in-memory resource either does not exist or the caller does not have access to it. Action: Correct the resource name or ensure that the caller has access to the resource.
X'0C' IFAINMRetCodeEnvErr	X'0C02' IFAINMSMFNotActive	Meaning: SMF is not active. Action: None.
X'10' IFAINMRetCodeFatal	Not applicable	Meaning: Internal error. Action: Contact the IBM Support Center.

IFAMDSC — Disconnect from an SMF in-memory resource

Call the IFAMDSC service to disconnect from an SMF in-memory resource.

Description

The IFAMDSC service disconnects from an SMF in-memory resource when the calling program no longer needs to request any more data via the IFAMGET service.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.

Programming requirements

A prior successful call to the IFAMCON service is required.

Authorization

A prior successful call to the IFAMCON service is required.

Restrictions

The disconnect request must be issued from the same address space as the connect request.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
Call IFAMDSC (
           DscParmBlock,
           rc,
           rsn);
```

For assembler programming, the stub routine for this callable service resides in SYS1.CSSLIB.

Parameters

The parameters are explained as follows:

DscParmBlock

A disconnect parameter block.

Direction: Input/output

Type: Disconnect parameter block

The disconnect parameter block has the following format and is mapped by the IFAZSYSP macro in SYS1.MACLIB:

Decimal offset	Length	Type	Field	Description
0	4	EBCDIC	ID	ID with value of DSPB
4	2	binary	Length	Total length of the parameter block
6	1	binary	Unused	Unused; must be zero
7	1	binary	Version	Parameter block version number (X'01')
8	16		Token	Token provided by the IFAMCON service

rc The return code from the service.

Direction: Output

Type: 4-word integer

rsn

The reason code from the service.

Direction: Output

Type: 4-word integer

ABEND codes

IFAMDSC might abnormally end with system completion code X'353'. See *z/OS MVS System Codes* for an explanation and programmer response.

Related services

- “IFAMCON — Connect to an SMF in-memory resource” on page 567
- “IFAMGET — Obtain data from an SMF in-memory resource” on page 573

Return and reason codes

When the service returns control to the caller, the **rc** parameter contains a hexadecimal return code and the **rsn** parameter contains a hexadecimal reason code, as listed in Table 103. The return code and reason code symbols are mapped by the IFARCINM macro.

Table 103. Return and reason codes for the IFAMDSC service

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'00' IFAINMRetCodeOK	X'0000' IFAINMRsnCodeOK	Meaning: Successful completion. Action: None.
X'04' IFAINMRetCodeWarn	X'0405' IFAINMGetInProgDsc	Meaning: There is currently a GET call in progress for this connection. The active GET call has been notified of the disconnection attempt. Action: Wait for the GET call to return before issuing another disconnect request.
X'04' IFAINMRetCodeWarn	X'0407' IFAINMDscInProgDsc	Meaning: There is currently a DISCONNECT call in progress for this connection. Action: None.
X'08' IFAINMRetCodeError	X'0801' IFAINMBadMode	Meaning: The caller is running in an incorrect mode for one or more of the following reasons: <ul style="list-style-type: none"> • The caller is not running in task mode. • The caller is in cross-memory mode. • The caller is holding a lock. Action: Change the program to run under a task in PASN = HASN mode with no locks held.
X'08' IFAINMRetCodeError	X'0802' IFAINMBadParmlist	Meaning: The parameter block is not accessible or has an incorrect format. Action: Correct the program to pass a valid parameter block.
X'08' IFAINMRetCodeError	X'0804' IFAINMBadConToken	Meaning: The caller is attempting to pass an invalid token in the parameter block. Action: Correct the program to provide the correct token that was returned on the IFAMCON call.
X'08' IFAINMRetCodeError	X'0805' IFAINMUnSupported	Meaning: The caller is attempting to pass unsupported options in the parameter block. Action: Initialize all unused fields in the parameter block to zero.

Table 103. Return and reason codes for the IFAMDSC service (continued)

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'0C' IFAINMRetCodeEnvErr	X'0C02' IFAINMSMFNotActive	Meaning: SMF is not active. Action: None.
X'0C' IFAINMRetCodeEnvErr	X'0C03' IFAINMObtainFailure	Meaning: SMF is unable to obtain storage to generate in-memory data in response to this request. Action: None.
X'10' IFAINMRetCodeFatal	Not applicable	Meaning: Internal error. Action: Contact the IBM Support Center.

IFAMGET — Obtain data from an SMF in-memory resource

Call the IFAMGET service to obtain data from an SMF in-memory resource.

Description

The IFAMGET service obtains real-time SMF data as it is being recorded to an in-memory resource. The service can immediately return to the caller when there are no records available, or it can wait for a new record to become available before returning.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN = HASN
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Programming requirements

The length of the caller-provided output buffer that is to hold the returned SMF records must be 32,768 bytes or larger.

Authorization

A prior successful call to the IFAMCON service is required.

Restrictions

None.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters on the CALL statement in the order shown.

IFAMGET

```
Call IFAMGET (  
    GetParmBlock,  
    rc,  
    rsn);
```

For assembler programming, the stub routine for this callable service resides in SYS1.CSSLIB.

Parameters

The parameters are explained as follows:

GetParmBlock

A get parameter block.

Direction: Input/output

Type: Get parameter block

The get parameter block has the following format and is mapped by the IFAZSYSP macro in SYS1.MACLIB:

Decimal offset	Length	Type	Field	Description
0	4	EBCDIC	ID	ID with value of GTPB
4	2	binary	Length	Total length of the get parameter block
6	1	binary	Unused	Unused; must be zero
7	1	binary	Version	Parameter block version number (X'01')
8	4	binary	Flags	The following flags are supported: Value Meaning X'80' Reserved; must be zero. X'40' Reserved; must be zero. X'20' Reserved; must be zero. X'10' If no record is available, return immediately to the caller.
12	4	binary	Reserved	Reserved; must be zero
16	16	binary	Token	Token provided by the IFAMCON service
32	4	binary	Buffer length	Length of the caller-provided output buffer Requirement: The buffer length must be 32,768 bytes or larger.
36	16	binary	Reserved	Reserved; must be zero

Decimal offset	Length	Type	Field	Description
52	4	binary	Returned length	Length of the data returned in the output buffer
56	8	binary	Buffer address	Pointer to the caller-provided output buffer that is to hold the returned SMF records

rc The return code from the service.

Direction: Output

Type: 4-word integer

rsn

The reason code from the service.

Direction: Output

Type: 4-word integer

ABEND codes

IFAMGET might abnormally end with system completion code X'353'. See *z/OS MVS System Codes* for an explanation and programmer response.

Related services

- “IFAMCON — Connect to an SMF in-memory resource” on page 567
- “IFAMDSC — Disconnect from an SMF in-memory resource” on page 570

Return and reason codes

When the service returns control to the caller, the **rc** parameter contains a hexadecimal return code and the **rsn** parameter contains a hexadecimal reason code, as listed in Table 104. The return code and reason code symbols are mapped by the IFARCINM macro.

Table 104. Return and reason codes for the IFAMGET service

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'00' IFAINMRetCodeOK	X'0000' IFAINMRsnCodeOK	Meaning: Successful completion. Action: None.
X'04' IFAINMRetCodeWarn	X'0401' IFAINMMissedData	Meaning: Records were skipped due to buffer re-use—that is, wrapping of the data in the in-memory resource. In this case, the output buffer might not contain a valid record. Action: The cursor is re-synched to the newest data on the next GET call. The application must be able to handle the missing data condition.

Table 104. Return and reason codes for the IFAMGET service (continued)

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'04' IFAINMRetCodeWarn	X'0402' IFAINMNoMoreData	Meaning: No records are available due to the removal of the in-memory resource from the configuration. In this case, the output buffer does not contain a valid record. Action: The caller should disconnect from the resource.
X'04' IFAINMRetCodeWarn	X'0403' IFAINMNoMoreDataTmp	Meaning: No records are available for this non-blocking request. In this case, the output buffer does not contain a valid record. Action: The caller can reinvoke the IFAMGET service to check for new records.
X'04' IFAINMRetCodeWarn	X'0404' IFAINMGetInProgGet	Meaning: A GET call is in progress for this connection. Action: Wait for the active GET call to complete before issuing another GET call.
X'04' IFAINMRetCodeWarn	X'0406' IFAINMDscInProgGet	Meaning: The caller attempted a GET call while a DISCONNECT call was in progress for this connection. When the active DISCONNECT call completes, the connection will no longer be active. Action: Reestablish a connection if you wish to perform a GET call.
X'04' IFAINMRetCodeWarn	X'0408' IFAINMGetForcedOut	Meaning: A request was made to disconnect an active connection while a GET call was in progress. The GET call ends, and subsequent GET calls cannot be issued for this connection. Action: Reestablish a connection if you wish to perform a GET call.
X'08' IFAINMRetCodeError	X'0801' IFAINMBadMode	Meaning: The caller is running in an incorrect mode for one or more of the following reasons: <ul style="list-style-type: none"> • The caller is in cross-memory mode. • The caller is holding a lock. Action: Change the program to run under a task or SRB in PASN = HASN mode with no locks held.
X'08' IFAINMRetCodeError	X'0802' IFAINMBadParmlist	Meaning: The parameter block is not accessible or has an incorrect format. Action: Correct the program to pass a valid parameter block.
X'08' IFAINMRetCodeError	X'0804' IFAINMBadConToken	Meaning: The caller is attempting to pass an invalid token in the parameter block. Action: Correct the program to provide the correct token that was returned on the IFAMCON call.
X'08' IFAINMRetCodeError	X'0805' IFAINMUnsupported	Meaning: The caller is attempting to pass unsupported options in the parameter block. Action: Initialize all unused fields in the parameter block to zero.
X'08' IFAINMRetCodeError	X'0806' IFAINMNotEnoughSpace	Meaning: There is not enough space in the output buffer to hold the returned record. Action: Call the IFAMGET service with an output buffer that is large enough to contain the record.
X'0C' IFAINMRetCodeEnvErr	X'0C02' IFAINMSMFNotActive	Meaning: SMF is not active. Action: None.

Table 104. Return and reason codes for the IFAMGET service (continued)

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'10' IFAINMRetCodeFatal	Not applicable	Meaning: Internal error. Action: Contact the IBM Support Center.

IFAMQRY — Query SMF in-memory resources

Call the IFAMQRY service to query the SMF in-memory resources that are available to the application.

Description

Your application can call the IFAMQRY service to determine which SMF in-memory resources are available. Only those in-memory resources that are available to this caller, as determined by SAF, access are returned.

Note: The returned data represents point-in-time information that is subject to change because of configuration changes before a call to IFAMCON is made. Results are determined based on the caller's access to the data.

Environment

The requirements for the caller are:

Requirement	Details
Minimum authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Programming requirements

None.

Authorization

None.

Restrictions

None.

Syntax

Write the call as shown in the following syntax diagram. You must code all parameters on the CALL statement in the order shown.

IFAMQRY

```
Call IFAMQRY (
                QryParmBlock,
                rc,
                rsn);
```

For assembler programming, the stub routine for this callable service resides in SYS1.CSSLIB.

Parameters

The parameters are explained as follows:

QryParmBlock

A query parameter block.

Direction: Input/output

Type: Query parameter block

The query parameter block has the following format and is mapped by the IFAZSYSP macro in SYS1.MACLIB:

Decimal offset	Length	Type	Field	Description
0	4	EBCDIC	ID	ID with value of QRPB
4	2	binary	Length	Total length of the query parameter block
6	1	binary	Unused	Unused; must be zero
7	1	binary	Version	Parameter block version number (X'01')
8	4	binary	Unused	Unused; must be zero
12	4	binary	Returned Imrs	Number of returned in-memory resources
16	4	binary	Buffer size	Size of the output buffer that is to contain the information for the returned in-memory resources
20	4	binary	Unused	Unused; must be zero
24	8	binary	Buffer address	Address of the output buffer that is to contain the information for the returned in-memory resources

Each in-memory resource returned in the output buffer (pointed to by the address in the buffer address field) has the following format:

Decimal offset	Length	Type	Field	Description
0	2	binary	Name length	Length of the name
2	26	EBCDIC	Name	Name of the in-memory resource, padded with blanks

Decimal offset	Length	Type	Field	Description
28	32	binary	Types	Bit mask (0 - 255 bit array) of SMF record types that are available from this in-memory resource
60	8	binary	Reserved	Reserved

rc The return code from the service.

Direction: Output

Type: 4-word integer

rsn

The reason code from the service.

Direction: Output

Type: 4-word integer

ABEND codes

IFAMQRY might abnormally end with system completion code X'353'. See *z/OS MVS System Codes* for an explanation and programmer response.

Related services

- “IFAMCON — Connect to an SMF in-memory resource” on page 567
- “IFAMGET — Obtain data from an SMF in-memory resource” on page 573
- “IFAMDSC — Disconnect from an SMF in-memory resource” on page 570

Return and reason codes

When the service returns control to the caller, the **rc** parameter contains a hexadecimal return code and the **rsn** parameter contains a hexadecimal reason code, as listed in Table 105. The return code and reason code symbols are mapped by the IFARCINM macro.

Table 105. Return and reason codes for the IFAMQRY service

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'00' IFAINMRetCodeOK	X'0000' IFAINMRsnCodeOK	Meaning: Successful completion. Action: None.
X'08' IFAINMRetCodeError	X'0801' IFAINMBadMode	Meaning: The caller is running in an incorrect mode for one or more of the following reasons: <ul style="list-style-type: none"> • The caller is not running in task mode. • The caller is in cross-memory mode. • The caller is holding a lock. Action: Change the program to run under a task in PASN = HASN mode with no locks held.
X'08' IFAINMRetCodeError	X'0802' IFAINMBadParmlist	Meaning: The parameter block is not accessible or has an incorrect format. Action: Correct the program to pass a valid parameter block.

IFAMQRY

Table 105. Return and reason codes for the IFAMQRY service (continued)

Return code Equate symbol	Reason code Equate symbol	Meaning and action
X'08' IFAINMRetCodeError	X'0805' IFAINMUnSupported	Meaning: The caller is attempting to pass unsupported options in the parameter block. Action: Initialize all unused fields in the parameter block to zero.
X'08' IFAINMRetCodeError	X'0808' IFAINMNotEnoughQrySp	Meaning: There is not enough space in the output buffer to hold all of the in-memory resources available to the caller. Action: Call the IFAMQRY service with an output buffer that is large enough to contain the data.
X'0C' IFAINMRetCodeEnvErr	X'0C02' IFAINMSMFNotActive	Meaning: SMF is not active. Action: None.
X'0C' IFAINMRetCodeEnvErr	X'0C03' IFAINMObtainFailure	Meaning: The system was unable to obtain storage to generate in-memory data in response to a request. Action: None.
X'10' IFAINMRetCodeFatal	Not applicable	Meaning: Internal error. Action: Contact the IBM Support Center.

Part 11. Appendixes

Appendix A. BCPii communication error reason codes

All BCPii API invocations can experience a communication failure when communicating between the BCPii address space and the support element of the targeted Central Processor Complex (CPC). The calling program receives the HWI_COMMUNICATION_ERROR (101 hexadecimal, 257 decimal) return code when this occurs. One of the output parameters from each service is a Diagnostic Area (referred to as the DiagArea). For the HWI_COMMUNICATION_ERROR return code, the Diag_Commerr field in the DiagArea contains a more descriptive return code from the BCPii communications transport to help pinpoint the cause of the failure.

The following table provides a partial list of the descriptive communication transport error return codes, along with a suggested action to take.

Return code, in hexadecimal (in decimal)	Description / suggested action
0-63 (0-99)	These return codes are documented in Appendix C (API Return Codes) in <i>System z Application Programming Interfaces</i> (SB10-7030). For a Diag_Commerr value of X'15' (21 decimal), this may signify a possible busy condition on the targeted SE. An application may choose to retry the request. Persistent failures with this return code should be reported to the IBM Support Center.
64-76 (100-118)	An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.
77 (119)	The BCPii transport rejected the particular request. Activate CTRACE with CTRACE option "ALL" and reissue the request. If the request failed again, turn off CTRACE, collect the SVCDUMP, and contact the IBM Support Center.
78-CF (120-207)	An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.
D0 (208)	The support element rejected the particular request. This could occur for any number of reasons including: the SE is busy, the SE is rebooting, etc. Consider retrying the request one or more times. If the problem persists, activate CTRACE with CTRACE option "ALL" and reissue the request. Then turn off CTRACE, collect the SVCDUMP, and contact the IBM Support Center.
D1-D3 (209-211)	An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.
D4 (212)	The support element rejected communication from BCPii, likely because the Cross partition authority was not granted on this support element.
E0 (224)	No response was received from the support element, after waiting for a considerable amount of time. BCPii times out the request. Check if connectivity to the support element is still there.
Greater than E0 (>224)	An internal error has likely occurred inside the BCPii transport code. Contact the IBM Support Center.

BCPii Communication Error Reason Codes

Appendix B. BCPii summary tables

The following summary tables show the objects that can be targeted for the BCPii functions:

- “HWIQUERY and HWISET”
- “HWICMD” on page 596
- “HWIEVENT” on page 598

For complete details of the BCPii APIs, see Chapter 19, “Base Control Program internal interface (BCPii),” on page 257.

HWIQUERY and HWISET

This table shows the BCPii HWIQUERY and HWISET attributes and the objects that can be targeted for each function. Note: The HWMCA attribute suffix refers to the 'HWMCA Object Attribute ID suffix' documented in *System z Application Programming Interfaces (SB10-7030-13)*.

Table 106. HWIQUERY and HWISET attributes

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	HWMCA attribute suffix
HWI_NAME 1 (1)	Name	V1R10		X	X		X	X	X	X	1.0
HWI_ERRSTAT 2 (2)	Status error (Y/N)	V1R10		X	X					X	7.0
HWI_BUYSYSTAT 3 (3)	Busy status (Y/N)	V1R10		X	X					X	8.0
HWI_MSGSTAT 4 (4)	Messages present (Y/N)	V1R10		X	X						9.0
HWI_OPERSTAT 5 (5)	Current status	V1R10		X	X						10.0
HWI_ACCSTAT 6 (6)	Acceptable status values	V1R10	X	X	X						11.0
HWI_APROF 7 (7)	Next reset activation profile name	V1R10	X	X	X						13.0
HWI_LUAPROF 8 (8)	Last used activation profile name	V1R10		X	X						14.0
HWI_OBJTYPE 9 (9)	Object type	V1R10		X	X	X	X	X	X	X	22.0
HWI_IMLMODE A (10)	IML mode	V1R10		X	X						12.0
HWI_IPADDR 17 (23)	Internet address (IPv4 format)	V1R10		X							15.0

HWIQUERY and HWISET attributes

Table 106. HWIQUERY and HWISET attributes (continued)

Attribute constant equates symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User- defined Image Group	HWMCA attribute suffix
HWI_SNAADDR 18 (24)	SNA address (netid.nau)	V1R10		X							16.0
HWI_MMODEL 19 (25)	Machine model	V1R10		X							17.0
HWI_MTYPE 1A (26)	Machine type	V1R10		X							18.0
HWI_MSERIAL 1B (27)	Machine serial	V1R10		X							19.0
HWI_CPCSERIAL 1C (28)	CPC serial number	V1R10		X							20.0
HWI_CPCID 1D (29)	CPC identifier	V1R10		X							21.0
HWI_RESERVEID 1E (30)	Name of application holding reserve	V1R10		X							44.0
HWI_SVCEREQD 1F (31)	Service required (Y/N)	V1R10		X							46.0
HWI_CBUINSTD 20 (32)	CBU installed (Y/N)	V1R10		X							32.0
HWI_CBUENABLD 21 (33)	CBU enabled (Y/N)	V1R10		X							48.0
HWI_CBUACTIVE 22 (34)	CBU activated (Y/N)	V1R10		X							33.0
HWI_CBUACTDT 23 (35)	CBU activation date	V1R10		X							34.0
HWI_CBUEXPDT 24 (36)	CBU expiration date	V1R10		X							35.0
HWI_CBUTESTAR 25 (37)	CBU test activations remaining	V1R10		X							36.0
HWI_CBUREALAV 26 (38)	Real CBU activation available (Y/N)	V1R10		X							37.0
HWI_PRUNTYPE 27 (39)	Processor running time type	V1R10	X	X			X				78.0
HWI_PRUNTIME 28 (40)	Processor running time	V1R10	X	X			X				79.0
HWI_PRUNTSEW 29 (41)	Processor loses its running time slice when in wait state (Y/N)	V1R10	X	X			X				80.0
HWI_OOCINST 2A (42)	On/Off on Demand installed (Y/N)	V1R10		X							87.0
HWI_OOCTACT 2B (43)	On/Off on Demand activated (Y/N)	V1R10		X							88.0

Table 106. HWIQUERY and HWISET attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	HWMCA attribute suffix
HWI_OOCENAB 2C (44)	On/Off on Demand enabled (Y/N)	V1R10		X							89.0
HWI_OOCADT 2D (45)	On/Off on Demand activation date	V1R10		X							90.0
HWI_PCPCSWM 2E (46)	Permanent CPC software model	V1R10		X							120.0
HWI_PPBPMSW 2F (47)	Permanent plus billable processor software model	V1R10		X							121.0
HWI_PPTPSW 30 (48)	Permanent plus (all) temporary processor software model	V1R10		X							122.0
HWI_PCPCMSU 31 (49)	CPC millions of service units (MSU) value	V1R10		X							123.0
HWI_PPBPMSU 32 (50)	Permanent plus billable processor MSU value	V1R10		X							124.0
HWI_PPTPMSU 33 (51)	Permanent plus (all) temporary processor MSU value	V1R10		X							125.0
HWI_NUMGPP 34 (52)	Number of general purpose processors	V1R10		X							126.0
HWI_NUMSAP 35 (53)	Number of service assist processors	V1R10		X							127.0
HWI_NUMIFAP 36 (54)	Number of integrated facility for applications (IFA) processors	V1R10		X							128.0
HWI_NUMIFLP 37 (55)	Number of integrated facility for Linux (IFL) processors	V1R10		X							129.0
HWI_NUMICFP 38 (56)	Number of internal coupling facility (ICF) processors	V1R10		X							130.0
HWI_NUMIIPP 39 (57)	Number of integrated information (IIP) processors	V1R10		X							131.0
HWI_NUMFLTYP 3A (58)	Number of defective (faulty) processors	V1R10		X							132.0
HWI_NUMSPARE 3B (59)	Number of spare processors	V1R10		X							133.0
HWI_NUMPENDP 3C (60)	Number of pending (activation) processors	V1R10		X							134.0
HWI_CAPCHGALLWD 3D (61)	Allow temporary capacity change (Y/N)	V1R10		X							149.0

HWIQUERY and HWISET attributes

Table 106. HWIQUERY and HWISET attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	HWMCA attribute suffix
HWI_DGRSTAT 3E (62)	Degraded status	V1R10		X							47.0
HWI_CURRPOWERMODE 3F (63)	Current processor power savings mode activated	V1R10		X							190.0
HWI_SUPPPPOWERMODE 40 (64)	Supported processor power savings modes available	V1R10		X							191.0
HWI_STPCONFIG 41 (65)	Server Timer Protocol (STP) configuration data	V1R12		X							165.0
HWI_NUMPGPP 42 (66)	Number of pending general purpose processors	V1R12		X							175.0
HWI_NUMPSAP 43 (67)	Number of pending service assist processors	V1R12		X							176.0
HWI_NUMPAAP 44 (68)	Number of pending Application Assist (AAP) processors	V1R12		X							177.0
HWI_NUMPIFLP 45 (69)	Number of pending Integrated Facility for Linux (IFL) processors	V1R12		X							178.0
HWI_NUMPICFP 46 (70)	Number of pending Internal Coupling Facility (ICF) processors	V1R12		X							179.0
HWI_NUMPIIPP 47 (71)	Number of pending Integrated Information (IIP) processors	V1R12		X							180.0
HWI_POWERMODE ALLOWED 48 (72)	Processor power savings mode allowed (Y/N)	V1R10		X							193.0
HWI_VERSION 49 (73)	CPC version number	V1R13		X							151.0
HWI_EC_MCL_INFO 4A (74)	XML string that describes the Engineering Change (EC) and Microcode Level (MCL) levels	V1R13		X							162.0
HWI_LIST_IP_ADDRESSES 4B (75)	All the IP addresses (in IPv4 and/or IPv6 format)	V1R13		X							161.0
HWI_AUTO_SWITCH_ENABLED 4C (76)	Automatic switching between primary and alternate support elements enabled (Y/N)	V1R13		X							163.0

HWIQUERY and HWISET attributes

Table 106. HWIQUERY and HWISET attributes (continued)

Attribute constant equates symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User- defined Image Group	HWMCA attribute suffix
HWI_CPCNAME 69 (105)	Parent (CPC) name	V1R10			X						2.0
HWI_OSNAME 6A (106)	Operating system name	V1R10			X						3.0
HWI_OSTYPE 6B (107)	SW operating system type (MVS, VM, LINUX, VSE, Z TPF EE)	V1R10			X						4.0
HWI_OSLEVEL 6C (108)	SW operating system level	V1R10			X						5.0
HWI_SYSPLEX 6D (109)	SW sysplex name	V1R10			X						6.0
HWI_CLUSTER 6E (110)	LPAR cluster name	V1R10			X						49.0
HWI_PARTITIONID 6F (111)	Partition ID	V1R10			X			X			51.0
HWI_DEFCAP 70 (112)	Current defined	V1R10	X		X			X			43.0
HWI_SGPIPW 71 (113)	Shared general processor initial processing weight	V1R10	X		X			X			30.0
HWI_SGPIWCAP 72 (114)	SGPIPW capped (Y/N)	V1R10	X		X			X			31.0
HWI_SGPPWMIN 73 (115)	Minimum SGPPW value	V1R10	X		X			X			38.0
HWI_SGPPWMAX 74 (116)	Maximum SGPPW value	V1R10	X		X			X			39.0
HWI_SGPPW 75 (117)	Current SGPPW value	V1R10			X						41.0
HWI_SGPPWCAP 76 (118)	SGPPW capped (Y/N)	V1R10			X						42.0
HWI_WLM 77 (119)	WLM allowed to change processing weight related attributes (Y/N)	V1R10	X		X			X			40.0
HWI_IFAIPW 78 (120)	Integrated facility for applications initial processing weight	V1R10	X		X			X			60.0
HWI_IFAIPWCAP 79 (121)	IFAIPW capped (Y/N)	V1R10	X		X			X			61.0
HWI_IFAPWMIN 7A (122)	Minimum IFAPW value	V1R10	X		X			X			62.0

HWIQUERY and HWISET attributes

Table 106. HWIQUERY and HWISET attributes (continued)

Attribute constant equates symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User- defined Image Group	HWMCA attribute suffix
HWI_IFAPWMAX 7B (123)	Maximum IFAPW value	V1R10	X		X			X			63.0
HWI_IFAPW 7C (124)	Current IFAPW value	V1R10			X						64.0
HWI_IFAPWCAP 7D (125)	IFAPW capped (Y/N)	V1R10			X						65.0
HWI_IFLIPW 7E (126)	Integrated facility for Linux initial processing weight	V1R10	X		X			X			66.0
HWI_IFLIPWCAP 7F (127)	IFLIPW capped (Y/N)	V1R10	X		X			X			67.0
HWI_IFLPWMIN 80 (128)	Minimum IFLPW value	V1R10	X		X			X			68.0
HWI_IFLPWMAX 81 (129)	Maximum IFLPW value	V1R10	X		X			X			69.0
HWI_IFLPW 82 (130)	Current IFLPW value	V1R10			X						70.0
HWI_IFLPWCAP 83 (131)	IFLPW capped (Y/N)	V1R10			X						71.0
HWI_ICFIPW 84 (132)	Internal coupling facility initial processing weight	V1R10	X		X			X			72.0
HWI_ICFIPWCAP 85 (133)	ICFIPW capped (Y/N)	V1R10	X		X			X			73.0
HWI_ICFPWMIN 86 (134)	Minimum ICFPW value	V1R10	X		X			X			74.0
HWI_ICFPWMAX 87 (135)	Maximum ICFPW value	V1R10	X		X			X			75.0
HWI_ICFPW 88 (136)	Current ICFPW value	V1R10			X						76.0
HWI_ICFPWCAP 89 (137)	ICFPW capped (Y/N)	V1R10			X						77.0
HWI_IIPIPW 8A (138)	Integrated information processors initial processing weight	V1R10	X		X			X			81.0
HWI_IIPIPWCAP 8B (139)	IIPIPW capped (Y/N)	V1R10	X		X			X			82.0
HWI_IIPWMIN 8C (140)	Minimum IIPW value	V1R10	X		X			X			83.0
HWI_IIPWMAX 8D (141)	Maximum IIPW value	V1R10	X		X			X			84.0

Table 106. HWIQUERY and HWISET attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	HWMCA attribute suffix
HWI_IIPPW 8E (142)	Current IIPPW value	V1R10			X						85.0
HWI_IIPWCAP 8F (143)	IIPPW capped (Y/N)	V1R10			X						86.0
HWI_IPLTOKEN 90 (144)	IPL token associated with the current IPL of the image	V1R11			X						164.0
HWI_PSW 91 (145)	PSW for each CP associated with the image	V1R11			X						150.0
HWI_GROUP_PROFILE_CAPACITY 92 (146)	Workload unit for the group profile associated with an image	V1R13	X		X						192.0
HWI_LAST_USED_LOADADDR 93 (147)	Last-used load address	V1R13			X						201.0
HWI_LAST_USED_LOADPARAM 94 (148)	Last-used load parameters	V1R13			X						202.0
HWI_RECID B7 (183)	Record ID	V1R10				X					135.0
HWI_RECTYPE B8 (184)	Record type	V1R10				X					136.0
HWI_ACTSTAT B9 (185)	Record activation status	V1R10				X					137.0
HWI_ACTDATE BA (186)	Record activation date	V1R10				X					138.0
HWI_EXPDATE BB (187)	Record expiration date	V1R10				X					139.0
HWI_ACTEXP BC (188)	Record activation expiration date	V1R10				X					140.0
HWI_MAXRADS BD (189)	Maximum real activation days	V1R10				X					141.0
HWI_MAXTADS BE (190)	Maximum test activation days	V1R10				X					142.0
HWI_REMRADS BF (191)	Remaining real activation days	V1R10				X					143.0
HWI_REMTADS C0 (192)	Remaining test activation days	V1R10				X					144.0
HWI_OOCODREC C1 (193)	Capacity record in XML format	V1R10				X					N/A
HWI_IOCDS C9 (201)	IOCDS	V1R11	X				X				27.0

HWIQUERY and HWISET attributes

Table 106. HWIQUERY and HWISET attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User- defined Image Group	HWMCA attribute suffix
HWI_IPL_ADDRESS CA (202)	IPL address	V1R11	X					X	X		28.0
HWI_IPL_PARM CB (203)	IPL parameter	V1R11	X					X	X		29.0
HWI_IPL_TYPE CC (204)	IPL type	V1R11	X					X	X		52.0
HWI_WW_ PORTNAME CD (205)	Worldwide port name	V1R11	X					X	X		53.0
HWI_BOOT_PGM_ SELECTOR CE (206)	Boot program selector	V1R11	X					X	X		54.0
HWI_LU_NUM CF (207)	Logical unit number value	V1R11	X					X	X		55.0
HWI_BOOTREC_ BLK_ADDR D0 (208)	Boot record logical block address	V1R11	X					X	X		56.0
HWI_OP SYS_ LOADPARAM D1 (209)	Operating system specific load parameter	V1R11	X					X	X		57.0
HWI_GROUP_PROF_ NAME D2 (210)	Name of group profile to be used for image	V1R11	X		X			X			93.0
HWI_LOAD_AT_ ACTIVATION D3 (211)	Image loaded (IPLed) after activation (Y/N)	V1R11	X					X			94.0
HWI_CENTRAL_ STOR D4 (212)	Initial amount of central storage (in MB) for image	V1R11	X					X			95.0
HWI_RES_CENTRAL_ STOR D5 (213)	Reserved amount of central storage (in MB) for image	V1R11	X					X			96.0
HWI_EXPANDED_ STOR D6 (214)	Initial amount of expanded storage (in MB) for image	V1R11	X					X			97.0
HWI_RES_ EXPANDED_STOR D7 (215)	Reserved amount of expanded storage (in MB) for image	V1R11	X					X			98.0
HWI_NUM_GPP D8 (216)	Number of dedicated general purpose processors for image	V1R11	X					X			99.0

Table 106. HWIQUERY and HWISET attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	HWMCA attribute suffix
HWI_NUM_RESGPP D9 (217)	Number of reserved dedicated general purpose processors for image	V1R11	X					X			100.0
HWI_NUM_IFA DA (218)	Number of dedicated integrated facility for applications (IFA) processors for image	V1R11	X					X			101.0
HWI_NUM_RESIFA DB (219)	Number of reserved dedicated integrated facility for applications (IFA) processors for image	V1R11	X					X			102.0
HWI_NUM_IFL DC (220)	Number of dedicated integrated facility for Linux (IFL) processors for image	V1R11	X					X			103.0
HWI_NUM_RESIFL DD (221)	Number of reserved dedicated integrated facility for Linux (IFL) processors for image	V1R11	X					X			104.0
HWI_NUM_ICF DE (222)	Number of dedicated internal coupling facility (ICF) processors for image	V1R11	X					X			105.0
HWI_NUM_RESICF DF (223)	Number of reserved dedicated internal coupling facility (ICF) processors for image	V1R11	X					X			106.0
HWI_NUM_ZIIP E0 (224)	Number of dedicated System z integration information processors (zIIPs) for image	V1R11	X					X			107.0
HWI_NUM_RESZIIP E1 (225)	Number of reserved dedicated System z integration information processors (zIIPs) for image	V1R11	X					X			108.0
HWI_NUM_SHARED_GPP E2 (226)	Number of shared general purpose processors for image	V1R11	X					X			109.0

HWIQUERY and HWISET attributes

Table 106. HWIQUERY and HWISET attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	HWMCA attribute suffix
HWI_NUM_RES_SHARED_GPP E3 (227)	Number of reserved shared general purpose processors for image	V1R11	X					X			110.0
HWI_NUM_SHARED_IFA E4 (228)	Number of shared integrated facility for applications (IFA) processors for image	V1R11	X					X			111.0
HWI_NUM_RES_SHARED_IFA E5 (229)	Number of reserved shared integrated facility for applications (IFA) processors for image	V1R11	X					X			112.0
HWI_NUM_SHARED_IFL E6 (230)	Number of shared integrated facility for Linux (IFL) processors for image	V1R11	X					X			113.0
HWI_NUM_RES_SHARED_IFL E7 (231)	Number of reserved shared integrated facility for Linux (IFL) processors for image	V1R11	X					X			114.0
HWI_NUM_SHARED_ICF E8 (232)	Number of shared internal coupling facility (ICF) processors for image	V1R11	X					X			115.0
HWI_NUM_RES_SHARED_ICF E9 (233)	Number of reserved shared internal coupling facility (ICF) processors for image	V1R11	X					X			116.0
HWI_NUM_SHARED_ZIIP EA (234)	Number of shared System z integrated information processors (zIIPs) for image	V1R11	X					X			117.0
HWI_NUM_RES_SHARED_ZIIP EB (235)	Number of reserved shared System z integrated information processors (zIIPs) for image	V1R11	X					X			118.0
HWI_BASIC_CPU_AUTH_COUNT_CNTL EC (236)	Basic CPU counter facility for the image enabled (Y/N)	V1R12	X					X			168.0
HWI_PROBSTATE_CPU_AUTH_COUNT_CNTL ED (237)	Problem state CPU counter facility for the image enabled (Y/N)	V1R12	X					X			169.0

Table 106. HWIQUERY and HWISET attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	HWMCA attribute suffix
HWI_CRYPTOACTIONIVITY_CPU_AUTH_COUNT_CNTL EE (238)	Crypto activity CPU counter facility for the image enabled (Y/N)	V1R12	X					X			170.0
HWI_EXTENDED_CPU_AUTH_COUNT_CNTL EF (239)	Extended CPU counter facility for the image enabled (Y/N)	V1R12	X					X			171.0
HWI_COPROCESSOR_CPU_AUTH_COUNT_CNTL F0 (240)	Coprocessor group CPU counter facility for the image enabled (Y/N)	V1R12	X					X			172.0
HWI_BASIC_CPU_SAMPLING_AUTH_CNTL F1 (241)	Basic CP CPU sampling facility for the image enabled (Y/N)	V1R12	X					X			173.0
HWI_APROF_STORE_STATUS F2 (242)	Store status selected (Y/N)	V1R11	X						X		166.0
HWI_APROF_LOADTYPE F3 (243)	Type of load requested	V1R11	X						X		167.0
HWI_PROFILE_DESCRIPTION F4 (244)	Activation profile description	V1R13	X					X			203.0
HWI_PROFILE_PARTITION_ID F5 (245)	Partition identifier for AProf	V1R13	X					X			51.0
HWI_OPERATING_MODE F6 (246)	Operating mode value for AProf	V1R13	X					X			204.0
HWI_CLOCK_TYPE F7 (247)	Clock type assignment (time source setting)	V1R13	X					X			205.0
HWI_TIME_OFFSET_DAYS F8 (248)	Number of days currently set as offset from external time source's TOD	V1R13	X					X			206.0
HWI_TIME_OFFSET_HOURS F9 (249)	Number of hours currently set as offset from external time source's TOD	V1R13	X					X			207.0
HWI_TIME_OFFSET_MINUTES FA (250)	Number of minutes currently set as offset from external time source's TOD	V1R13	X					X			208.0
HWI_TIME_OFFSET_INCREASE FB (251)	Local time zone: TRUE means east of GMT; FALSE means west of GMT	V1R13	X					X			209.0

HWIQUERY and HWISET attributes

Table 106. HWIQUERY and HWISET attributes (continued)

Attribute constant equate symbol with hexadecimal and (decimal) values	Description	Starting z/OS release	Settable using HWISET	CPC	Image	CapRec	Reset AProf	Image AProf	Load AProf	User-defined Image Group	HWMCA attribute suffix
HWI_LICCC_VALIDATION_ENABLED FC (252)	Activation profile must conform to the current LICCC configuration (Y/N)	V1R13	X					X			210.0
HWI_GLOBAL_PERFORMANCE_DATA_CONTROL FD (253)	LPAR can be used to view processing unit activity data for all other LPARs on the same CPC (Y/N)	V1R13	X					X			211.0
HWI_IO_CONFIGURATION_CONTROL FE (254)	LPAR can be used to read and write any IOCDs (Y/N)	V1R13	X					X			212.0
HWI_CROSS_PARTITION_AUTHORITY FF (255)	LPAR can be used to issue instructions that reset or deactivate other LPARs (Y/N)	V1R13						X			213.0
HWI_LOGICAL_PARTITION_ISOLATION 100 (256)	Re-configurable channel paths assigned to LPAR are reserved for its exclusive use (Y/N)	V1R13	X					X			214.0

HWICMD

This table shows the BCPii HWICMD types and the objects that can be targeted for each command.

Table 107. HWICMD types

Command type / Constant with hexadecimal and (decimal) values	Description	Starting z/OS release	CPC	Image	User-defined Image Group
HWI_CMD_ACTIVATE 1 (1)	Activate target object	• CPC and image: V1R10 • User-defined image group: V1R13	X	X	X
HWI_CMD_DEACTIVATE 2 (2)	Deactivate target object	• CPC and image: V1R10 • User-defined image group: V1R13	X	X	X
HWI_CMD_HWMMSG 3 (3)	Resend all hardware messages or delete one hardware message	V1R10	X		
HWI_CMD_CBU 4 (4)	Activate or deactivate capacity backup	V1R10	X		
HWI_CMD_OOCOD 5 (5)	Activate or deactivate On/Off Capacity on Demand	V1R10	X		

Table 107. HWICMD types (continued)

Command type / Constant with hexadecimal and (decimal) values	Description	Starting z/OS release	CPC	Image	User-defined Image Group
HWI_CMD_PROFILE 6 (6)	Import or export activation profiles	V1R10	X		
HWI_CMD_RESERVE 7 (7)	Add or delete a reserve for an application	V1R10	X		
HWI_CMD_SYSRESET 8 (8)	Reset target object	<ul style="list-style-type: none"> Image: V1R10 User-defined image group: V1R13 		X	X
HWI_CMD_START 9 (9)	Start all CPs on target object	<ul style="list-style-type: none"> Image: V1R10 User-defined image group: V1R13 		X	X
HWI_CMD_STOP A (10)	Stop all CPs on target object	<ul style="list-style-type: none"> Image: V1R10 User-defined image group: V1R13 		X	X
HWI_CMD_PSWRESTART B (11)	Restart one CP	<ul style="list-style-type: none"> Image: V1R10 User-defined image group: V1R13 		X	X
HWI_CMD_OSCMD C (12)	Issue an operating system command	V1R10		X	
HWI_CMD_LOAD D (13)	IPL operating system or systems	<ul style="list-style-type: none"> Image: V1R10 User-defined image group: V1R13 		X	X
HWI_CMD_TEMPCAP E (14)	Add or remove temporary capacity	V1R10	X		
HWI_CMD_SYSRESET_IPLT F (15)	Reset an image if the IPL token matches the specified IPLT	V1R11		X	
HWI_CMD_ACTIVATE_WITH_ACTPROF 10 (16)	Activate using the specified activation profile	V1R11	X	X	
HWI_CMD_POWER_CONTROL 11 (17)	Specify power control characteristics	V1R10	X		
HWI_CMD_SCSI_LOAD 12 (18)	IPL Linux operating system or systems	<ul style="list-style-type: none"> Image: V1R12 User-defined image group: V1R13 		X	X
HWI_CMD_SCSI_DUMP 13 (19)	Dump a Linux operating system	V1R12		X	
HWI_CMD_SYSPLEX_TIME_SWAP_CTS 14 (20)	Swap the role of current time server (CTS) in a configured STP-only coordinated timing network (CTN) from preferred time server to backup time server or vice versa	V1R13	X		

HWICMD attributes

Table 107. HWICMD types (continued)

Command type / Constant with hexadecimal and (decimal) values	Description	Starting z/OS release	CPC	Image	User-defined Image Group
HWI_CMD_SYSPLEX_TIME_SET_STP_CONFIG 15 (21)	Set the configuration for an STP-only coordinated timing network (CTN)	V1R13	X		
HWI_CMD_SYSPLEX_TIME_CHANGE_STP_ONLY_CTN 16 (22)	Change the STP_ID portion of the CTN ID for an entire STP-only coordinated timing network (CTN)	V1R13	X		
HWI_CMD_SYSPLEX_TIME_JOIN_STP_ONLY_CTN 17 (23)	Allow a CPC to join an STP-only coordinated timing network (CTN)	V1R13	X		
HWI_CMD_SYSPLEX_TIME_LEAVE_STP_ONLY_CTN 18 (24)	Remove a CPC from an STP-only coordinated timing network (CTN)	V1R13	X		

HWIEVENT

This table shows the BCPii HWIEVENT types and the objects that can be registered or unregistered for each event.

Table 108. HWIEVENT types

Event ID / Bit position in structure specified (non-REXX)	Description	Starting z/OS release	CPC	Image
Hwi_Event_CmdResp 97	Notice of command completion from the SE	V1R10	X	X
Hwi_Event_StatusChg 98	Object status change	V1R10	X	X
Hwi_Event_NameChg 99	Object name change	V1R10	X	X
Hwi_Event_ActProfChg 100	Object has changed associated activation profile	V1R10	X	X
Hwi_Event_ObjCreate 101	New object has been defined	V1R10	X	X
Hwi_Event_ObjDestroy 102	Object has been undefined	V1R10	X	X
Hwi_Event_ObjException 103	Object has entered into or out of an exception state	V1R10	X	X
Hwi_Event_ApplStarted 104	Console application has started	V1R10	X	
Hwi_Event_ApplEnded 105	Console application is ending	V1R10	X	

Table 108. HWIEVENT types (continued)

Event ID / Bit position in structure specified (non-REXX)	Description	Starting z/OS release	CPC	Image
Hwi_Event_HwMsg 106	Hardware message associated has been issued	V1R10	X	
Hwi_Event_HwMsgDel 107	Hardware message has been deleted	V1R10	X	
Hwi_Event_SecurityEvent 108	Security event has been logged	V1R10	X	
Hwi_Event_CapacityChg 109	Processing capacity has changed in some manner	V1R10	X	
Hwi_Event_CapacityRecord 110	A change has occurred to a temporary capacity record	V1R10	X	
Hwi_Event_OpSysMsg 111	Operating system message has been issued	V1R10		X
Hwi_Event_HwCommError 112	Hardware communication error received	V1R10	X	
Hwi_Event_BCPiIStatus 113	BCPii address space has stopped or started	V1R10		X
Hwi_Event_DisabledWait 114	An image has entered a disabled wait state	V1R10		X
Hwi_Event_PowerChange 115	Power characteristic or characteristics have changed	V1R10	X	

HWIEVENT attributes

Appendix C. General use C/C++ header files

Programming Interface Information

C/C++ header files are shipped in z/OS V1R4 SYS1.SAMPLIB. These header files are analogous to traditional z/OS MVS mapping macros and are provided for general use. The following table lists the members and describes the interface. Descriptions of the data areas referenced can be found in *z/OS MVS Data Areas* in the z/OS Internet library (<http://www.ibm.com/systems/z/os/zos/bkserv/>).

Member	Description
BLSCADPL	Describes same data areas as assembler macro BLSABDPL. Depends on BLSCDESC.
BLSCADSY	Describes same data areas as assembler macro BLSADSY.
BLSCCBSP	Describes same data areas as assembler macro BLSACBSP. Depends on BLSCDESC.
BLSCDESC	Describes same data areas as assembler macros BLSRDATC, BLSRDATS, BLSRDATT, BLSRESSY, and BLSRSASY. Many of the other members require that this header file be included before they are included.
BLSCDRPX	Describes same data areas as assembler macro BLSRDRPX. Depends on BLSCDESC.
BLSCNAMP	Describes same data areas as assembler macro BLSRNAMP. Depends on BLSCDESC.
BLSCPCQE	Describes same data areas as assembler macro BLSRPCQE. Depends on BLSCDESC.
BLSCPPR2	Describes same data areas as assembler macro BLSUPPR2.
BLSCPWHS	Describes same data areas as assembler macro BLSRPWHS. Depends on BLSCDESC.
BLSCXMSP	Describes same data areas as assembler macro BLSRXMSP. Depends on BLSCDESC.
BLSCXSSP	Describes same data areas as assembler macro BLSRXSSP. Depends on BLSCDESC.

End Programming Interface Information

Appendix D. 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 V2R2 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.

Additional notices

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates 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.

Permission Notice

This book includes information about certain callable service stub and linkage-assist (stub) routines contained in specific data sets that are intended to be bound or link-edited with code and run on z/OS systems. In connection with your authorized use of z/OS, you may bind or link-edit these stubs into your modules and distribute your modules with the included stubs for the purposes of developing, using, marketing and distributing programs conforming to the documented programming interfaces for z/OS, provided that each stub is included in its entirety, including any IBM copyright statements. These stubs have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply the reliability, serviceability, or function of these stub programs. The stub referred to in this book is contained in the following data set:

- SYS1.CSSLIB

Programming interface information

This information is intended to help the customer to write applications that use operating system services. This information documents general-use programming interface and associated guidance information provided by z/OS.

General-use programming interfaces allow the customer to write programs that obtain the services of z/OS.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.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>).

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Glossary

This glossary defines technical terms and abbreviations used in z/OS MVS documentation. If you do not find the term you are looking for, view IBM Glossary of Computing Terms, located at: <http://www.ibm.com/ibm/terminology>

data object

A VSAM linear data set.

A storage area, outside the user's storage, that window services defines as a temporary object.

data-in-virtual

An MVS facility that enables a user to access a data object as though that data object resided in the user's storage.

gap

The grouping of consecutive bytes that the program repeatedly skips over. When a reference pattern has a gap, gaps and reference units alternate throughout the data area. See also *reference pattern* and *reference unit*.

hiperspace

A range of up to two gigabytes of virtual storage that a program can use like a buffer.

linear data set

A type of VSAM data set where data is stored as a linear string of bytes.

mapping

A process where window services makes a data object or part of a data object accessible to a user program through a scroll area or through a window.

object See data object.

permanent data object

A virtual storage access method (VSAM)

linear data set that resides on DASD (also called a data-in-virtual object).

reference pattern

The order in which a program's instructions process a data structure, such as an array. A reference pattern can be sequential or random and can contain gaps.

reference unit

A grouping of consecutive bytes that the program references. If the reference pattern has a gap, the reference unit is the grouping of bytes between gaps; gaps and reference units alternate throughout the data area. If the reference pattern does not have gaps, the reference unit is a logical grouping according to the structure of the data.

scroll area

An area of expanded storage that window services obtains. For a permanent object, window services maps a window to the scroll area and maps the scroll area to the permanent data object. You can use the scroll area to make interim changes to a permanent data object. For a temporary data object, the scroll area is the data object. Window services maps the window to the scroll area.

scrolling

A process where window services saves changes that a user has made in a window. For a permanent data object, window services saves the changes in the scroll area, without updating the permanent object. For a temporary object, window services updates the temporary object.

temporary data object

An area of expanded storage that window services provides for use by your program. You can use this storage to hold temporary data instead of using a DASD

workfile. Window services provides no means for you to save a temporary data object.

VSAM

Virtual storage access method.

window

An area in the user's storage where the user can view or change data in a data object that window services has made available.

Index

A

- access to a data object
 - temporary object 9
- access to an object
 - terminating 22
- accessibility 603
 - contact IBM 603
 - features 603
- ADA programming language
 - example using window services 45
- application
 - in resource recovery 127
- application_backout_UR call 132
 - return and reason codes 135
 - syntax 134
- application_commit_UR call 136
 - return and reason codes 139
 - syntax 138
- assistive technologies 603
- authorized interfaces for zEDC 200, 207, 208, 221

B

- back out changes to protected resources 132
- BCPii REXX execs, setting up access to 268
- BCPii REXX restrictions 273
- BCPii REXX support 272, 273, 276, 277
- blocks of an object
 - definition 3
 - size 3

C

- C programming language
 - call syntax for latch manager services 109
 - example of reference pattern services 91
 - example using window services 50
- call statements for latch manager services 109
- call statements for reference pattern services 87
- call syntax
 - for latch manager service 109
- CEA TSO/E address space services
 - CEATsoRequest API 151
 - components 143
 - diagnostic codes 167
 - invoking 151
 - overview 143
 - prerequisites 143
 - reason codes 163
 - request types 157
 - CeaTsoAttn 158
 - CeaTsoEnd 159
 - CeaTsoPing 160

- CEA TSO/E address space services
 - (*continued*)
 - request types (*continued*)
 - CeaTsoQuery 160
 - CeaTsoQueryApp 161
 - CeaTsoStart 157
 - requirements for callers 157
 - return codes 163
 - TSO/E address spaces 143
- changed data in an object
 - refreshing 19
- client web enablement toolkit 417
 - HTTP/HTTPS protocol enabler 497
 - JSON parser 419
- COBOL programming language
 - call syntax for latch manager services 109
 - example using reference pattern services 94
 - example using window services 53
- commit changes to protected resources 136
- commit protocol, two-phase 128
- compression service
 - memory registration 212
 - Rendezvous 208, 211
 - single compression request 215
 - unregister memory 214
 - unrendezvous 220
- contact
 - z/OS 603
- CPC names
 - dynamic modification of 265
- CSRIDAC callable service 27
- CSRIRP callable service 87
 - example 83
- CSRL16J callable service
 - entry characteristics for the target routine 231
 - freeing dynamic storage for the target routine 232
 - parameter description 231
 - programming requirements 232
 - return codes 235
 - syntax 231
- CSRREFR callable service 31
- CSRRRP callable service 89
- CSRSAVE callable service 34
- CSRSCOT callable service 36
- CSRSIC include file 243
- CSRVIEW callable service 39

D

- data compression 199, 200
- data object 14
 - mapping 3
 - obtaining access 12
 - structure 3
- data to be viewed
 - identifying 16

- data-in-virtual object 3
- DFP requirement for window services 13

E

- examples
 - data object mapped to a window 4
 - structure of a data object 4

F

- FORTTRAN programming language
 - call syntax for latch manager services 109
 - example using reference pattern services 98
 - example using window services 57
- FPZ4ABC 215
- FPZ4DMR 214
- FPZ4PRB 211
- FPZ4RMR 212
- FPZ4RZV 208
- FPZ4URZ 220

G

- gap in reference pattern services
 - defining 76
 - definition 76
- glossary of terms 611

H

- HTTP/HTTPS enabler
 - callable services 508
 - options 552
- HTTP/HTTPS protocol enabler 497
 - availability 499
 - elements of 498
 - linkage 499, 500, 501
 - programming considerations 499, 501
 - environment 501
 - problem determination 503
 - recovery 504
 - security 502
 - programming examples 508
 - syntax 499, 500
- HWIREXX
 - return codes 274
 - setting up access to 268
- HWTCNST 428, 509
- HWTHCONN 510
- HWTHDISC 515
- HWTHINIT 520
- HWTHRQST 524
- HWTHRSET 530
- HWTHSET 535

HWTHSLST 541
 HWTHTERM 547
 HWTJCREN 429
 HWTJESCT 438
 HWTJGAEN 439
 HWTJGBOV 443
 HWTJGJST 447
 HWTJGNUE 451
 HWTJGNUV 455
 HWTJGOEN 460
 HWTJGVAL 466
 HWTJINIT 470
 HWTJPARS 474
 HWTJSERI 480
 HWTJSRCH 485
 HWTJTERM 492

I

identifying data object 12
 IEAAFFN callable service
 parameter descriptions 227
 purpose 227
 requirements 228
 restrictions and limitations 228
 return codes 228
 syntax 227
 IFAMCON 567
 IFAMDSC 570
 IFAMGET 573
 IFAMQRY 577
 interim changes to a permanent object
 saving 18
 ISGLCRT callable service
 syntax 110
 ISGLOBT callable service
 syntax 114
 ISGLPBA callable service
 syntax 122
 ISGLPRG callable service
 syntax 120
 ISGLREL callable service
 syntax 117
 ISV-provided REXX programming
 restrictions 277
 ISV-provided REXX support 277

J

JavaScript Object Notation (JSON) 419
 JSON parser 419
 availability 421
 callable services 426
 elements of 420
 linkage 421, 422, 423
 programming considerations 421,
 423
 syntax 421, 422

K

keyboard
 navigation 603
 PF keys 603
 shortcut keys 603

L

latch manager services
 ISGLCRT callable service
 syntax 110
 ISGLOBT callable service
 syntax 114
 ISGLPBA callable service
 syntax 122
 ISGLPRG callable service
 syntax 120
 ISGLREL callable service
 syntax 117

M

multiple views of an object
 defining 17

N

navigation
 keyboard 603
 Notices 607

P

Pascal programming language
 example using window services 61,
 101
 permanent object
 definition 3
 maximum size 3
 relationship to a data-in-virtual
 object 3
 structure 3
 PL/I programming language
 call syntax for latch manager
 services 109
 example using window services 65
 processor affinity 227
 protected
 resource 127

R

reference information 87, 109
 reference pattern services
 coding examples 91
 C programming language 91
 COBOL programming
 language 94
 FORTRAN programming
 language 98
 Pascal programming
 language 101
 overview 73
 use with data window services 15
 using 79
 reference unit in reference pattern
 services
 choosing 76
 definition 76
 REPLACE option for a window 15
 resource
 process for protecting 128

resource (*continued*)
 protecting 127
 protection on multiple systems 131
 requesting protection 131
 resource manager
 in resource recovery 127
 resource recovery
 distributed 131
 process 128
 programs 127
 requesting 131
 service 132, 136
 RETAIN option for a window 15
 REXX programming language
 call syntax for latch manager
 services 109
 REXX restrictions 273
 REXX support 272, 273
 RRS
 application_backout_UR call 132
 application_commit_UR call 136
 as sync-point manager 127

S

sending comments to IBM xiii
 shortcut keys 603
 size of an object
 extending 17
 SMF services 565
 SMF real-time interface 567
 SMS requirement for window
 services 13
 structure of a data object 3
 summary of changes xv, xvi
 as updated March 2014 xvii
 as updated September 2014 xvii
 z/OS V2R2 xvi
 Summary of changes xvii
 sync-point manager
 in resource recovery 127

T

temporary object
 definition 3
 functions supported 9
 maximum size 3
 overview of supported functions 9
 structure 3
 terminology 611
 transferring control to another routine
 CSRL16J 231
 TSO/E REXX programming
 restrictions 277
 TSO/E REXX support 276
 two-phase commit protocol 128

U

UR (unit of recovery)
 backing out 132
 committing 136
 user interface
 ISPF 603
 TSO/E 603

using protected resources 127

V

view of an object
terminating 20

W

ways that window services can map an
object 5
what window services provides 4
window
definition 3
use 3
window services 11
call statements 23
COBOL programming language 53
coding examples 45, 53
ADA programming language 45
C programming language 50
FORTRAN programming
language 57
Pascal programming language 61
PL/I programming language 65
functions provided 4
handling abends 22
handling return codes 22
reference information 23
services provided 4
ways to map an object 5
window services overview 3

Z

z/OS client web enablement toolkit
See client web enablement toolkit
zEDC 199, 200, 223
zEDC Express 199, 200
zEnterprise Data Compression
(zEDC) 199, 200
requirements 200
zlib for zEDC 200, 203, 204, 205, 206



Product Number: 5650-ZOS

Printed in USA

SA23-1377-08

