Distributed File Service zFS
Administration

Version 2 Release 1
This edition applies to Version 2 Release 1 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

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 document ...................... xi
How this document is organized .......... xi
Conventions used in this document ...... xi
z/OS information ........................ xii
References to DFS information ........ xii

How to send your comments to IBM xiii
If you have a technical problem ......... xiii

z/OS Version 2 Release 1 summary of changes ............. xv

Part 1. zFS administration guide ......... 1

Chapter 1. z/OS File System overview .... 3
Features ................................... 3
Terminology and concepts ................. 4
What's new or changed for zFS in z/OS V2R1 . 8

Chapter 2. Installing and configuring zFS ............. 11
zFS installation and configuration steps .... 11
Applying required APARs for z/OS V2R1 ... 14
Specifying zFS file systems as sysplex-aware . 15
Using zFS read/write sysplex-aware file systems 15
Changing the sysplex-awareness of a mounted zFS read/write file system .... 17

Chapter 3. Managing zFS processes ......... 19
Starting zFS .................................. 19
Stopping zFS .................................. 19
Determining file system status .......... 20

Chapter 4. Creating and managing zFS file systems using compatibility mode aggregates ............. 21
Creating a compatibility mode aggregate ... 21
Using version 1.5 aggregates and extended (v5) directories .......... 24
Creating a version 1.5 aggregate ......... 24
Converting an existing aggregate to version 1.5 . 25
Converting an existing v4 directory to an extended (v5) directory .......... 26
Guidelines for v4 to v5 conversion .......... 27
Growing a compatibility mode aggregate .... 27
Dynamically growing a compatibility mode aggregate .... 28
Creating a multi-volume compatibility mode aggregate .......... 29
Adding a volume to a compatibility mode aggregate ...... 31
Increasing the size of a compatibility mode aggregate .......... 31
Copying each file and directory of the aggregate to a larger data set .... 31
Copying the physical blocks of the aggregate to a larger data set .... 33
Decreasing the size of a compatibility mode aggregate .......... 36
Renaming or deleting a compatibility mode aggregate .......... 37
Changing zFS attributes on a mounted zFS compatibility mode file system ........ 38
Unmounting zFS file systems before copying or moving .......... 39
Understanding zFS disk space allocation ........ 40
How data is stored on systems before z/OS V1R13 ........ 42
Sharing zFS data in a non-shared file system sysplex .... 43
Minimum and maximum file system sizes .......... 44
Version 1.5 aggregates .......................... 44
Version 1.4 aggregates .......................... 44
v4 directory considerations .......... 45

Chapter 5. Using zFS in a shared file system environment ............. 51
Overview of the shared file system environment .... 51
Read-only mounted file systems .......... 51
zFS support for read/write non-sysplex aware mounted file systems .... 52
zFS support for read/write sysplex-aware mounted file systems .... 53
zFS support for read/write file systems with different levels of sysplex-awareness .... 53
zFS enhanced sysplex-aware support .......... 55
zFS ownership versus z/OS UNIX ownership of file systems .......... 55
Determining the file system owner .......... 56
When is the z/OS UNIX owner important? .... 58
Dynamic movement of the zFS owner .......... 59
Using zFS in a shared file system environment .... 60

Chapter 6. Performing a backup of zFS 63
Back up a zFS aggregate .................. 64
Restoring an aggregate with DFSMSdss logical restore .......... 64

Chapter 7. Migrating data from HFS to zFS ............. 67
Using the z/OS HFS to zFS migration tool .......... 67
Using the z/OS UNIX pax command .......... 67
Using an intermediate archive file .......... 68
Without using an intermediate archive file .... 68
### Chapter 8. Performance and debugging

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance tuning.</td>
<td>69</td>
</tr>
<tr>
<td>Total cache size</td>
<td>69</td>
</tr>
<tr>
<td>Metadata cache</td>
<td>70</td>
</tr>
<tr>
<td>Transaction cache</td>
<td>71</td>
</tr>
<tr>
<td>Vnode cache</td>
<td>71</td>
</tr>
<tr>
<td>User file and client file cache</td>
<td>71</td>
</tr>
<tr>
<td>Log files</td>
<td>72</td>
</tr>
<tr>
<td>Log file cache</td>
<td>72</td>
</tr>
<tr>
<td>Fixed storage</td>
<td>72</td>
</tr>
<tr>
<td>I/O balancing</td>
<td>73</td>
</tr>
<tr>
<td>Monitoring zFS performance</td>
<td>73</td>
</tr>
<tr>
<td>Resetting performance monitoring data</td>
<td>74</td>
</tr>
<tr>
<td>Sample zFS QUERY reports</td>
<td>74</td>
</tr>
<tr>
<td>Debugging aids for zFS</td>
<td>98</td>
</tr>
<tr>
<td>Overview of trace options for zFS</td>
<td>98</td>
</tr>
<tr>
<td>Understanding the salvager utility</td>
<td>99</td>
</tr>
<tr>
<td>Understanding zFS dumps</td>
<td>101</td>
</tr>
<tr>
<td>Determining the XCF protocol interface level</td>
<td>102</td>
</tr>
<tr>
<td>Saving initialization messages in a data set</td>
<td>102</td>
</tr>
<tr>
<td>Determining service levels</td>
<td>102</td>
</tr>
<tr>
<td>Understanding namespace validation and correction</td>
<td>103</td>
</tr>
<tr>
<td>Understanding delays and hangs in zFS using the zFS hang detector</td>
<td>104</td>
</tr>
<tr>
<td>Identifying storage shortages in zFS</td>
<td>109</td>
</tr>
<tr>
<td>Diagnosing disabled aggregates</td>
<td>110</td>
</tr>
<tr>
<td>Handling disabled aggregates</td>
<td>110</td>
</tr>
</tbody>
</table>

### Chapter 9. Overview of the zFS audit identifier

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabling the zFS auditid</td>
<td>113</td>
</tr>
</tbody>
</table>

### Part 2. zFS administration reference

<table>
<thead>
<tr>
<th>Chapter 10. z/OS system commands</th>
<th>117</th>
</tr>
</thead>
<tbody>
<tr>
<td>modify zfs process</td>
<td>119</td>
</tr>
<tr>
<td>setomvs reset</td>
<td>122</td>
</tr>
</tbody>
</table>

### Chapter 11. zFS commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ioeagfmt</td>
<td>125</td>
</tr>
<tr>
<td>ioeasplv</td>
<td>129</td>
</tr>
<tr>
<td>ioeufsult</td>
<td>134</td>
</tr>
<tr>
<td>ioeufsult converttov4</td>
<td>135</td>
</tr>
<tr>
<td>ioeufsult converttov5</td>
<td>137</td>
</tr>
<tr>
<td>ioeufsult format</td>
<td>139</td>
</tr>
<tr>
<td>ioeufsult salvage</td>
<td>142</td>
</tr>
<tr>
<td>MOUNT</td>
<td>146</td>
</tr>
<tr>
<td>zfsadm</td>
<td>149</td>
</tr>
<tr>
<td>zfsadm aggrinfo</td>
<td>153</td>
</tr>
<tr>
<td>zfsadm apropos</td>
<td>155</td>
</tr>
<tr>
<td>zfsadm attach</td>
<td>156</td>
</tr>
<tr>
<td>zfsadm config</td>
<td>159</td>
</tr>
<tr>
<td>zfsadm configquery</td>
<td>162</td>
</tr>
<tr>
<td>zfsadm convert</td>
<td>165</td>
</tr>
<tr>
<td>zfsadm define</td>
<td>168</td>
</tr>
<tr>
<td>zfsadm delete</td>
<td>170</td>
</tr>
<tr>
<td>zfsadm detach</td>
<td>171</td>
</tr>
<tr>
<td>zfsadm fileinfo</td>
<td>173</td>
</tr>
<tr>
<td>zfsadm format</td>
<td>178</td>
</tr>
<tr>
<td>zfsadm grow</td>
<td>180</td>
</tr>
<tr>
<td>zfsadm help</td>
<td>181</td>
</tr>
<tr>
<td>zfsadm lsaggr</td>
<td>183</td>
</tr>
<tr>
<td>zfsadm lsfs</td>
<td>184</td>
</tr>
<tr>
<td>zfsadm lssys</td>
<td>186</td>
</tr>
<tr>
<td>zfsadm query</td>
<td>186</td>
</tr>
<tr>
<td>zfsadm quiesce</td>
<td>188</td>
</tr>
<tr>
<td>zfsadm setauditfd</td>
<td>189</td>
</tr>
<tr>
<td>zfsadm unquiesce</td>
<td>191</td>
</tr>
</tbody>
</table>

### Chapter 12. The zFS configuration options file (IOEPRMxx or IOEFSRPM)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOEFSRPM</td>
<td>193</td>
</tr>
</tbody>
</table>

### Chapter 13. zFS application programming interfaces

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>pfctl (BPX1PCT)</td>
<td>207</td>
</tr>
<tr>
<td>Attach Aggregate</td>
<td>211</td>
</tr>
<tr>
<td>Define Aggregate</td>
<td>213</td>
</tr>
<tr>
<td>Delete File System</td>
<td>216</td>
</tr>
<tr>
<td>Detach Aggregate</td>
<td>220</td>
</tr>
<tr>
<td>Format Aggregate</td>
<td>222</td>
</tr>
<tr>
<td>Grow Aggregate</td>
<td>225</td>
</tr>
<tr>
<td>List Aggregate Status</td>
<td>228</td>
</tr>
<tr>
<td>List Aggregate Status (Version 2)</td>
<td>230</td>
</tr>
<tr>
<td>List Attached Aggregate Names</td>
<td>238</td>
</tr>
<tr>
<td>List Attached Aggregate Names (Version 2)</td>
<td>241</td>
</tr>
<tr>
<td>List File Information</td>
<td>244</td>
</tr>
<tr>
<td>List File System Names</td>
<td>249</td>
</tr>
<tr>
<td>List File System Names (Version 2)</td>
<td>252</td>
</tr>
<tr>
<td>List File System Status</td>
<td>255</td>
</tr>
<tr>
<td>List Systems</td>
<td>263</td>
</tr>
<tr>
<td>Query Config Option</td>
<td>265</td>
</tr>
<tr>
<td>Quiesce Aggregate</td>
<td>269</td>
</tr>
<tr>
<td>Reset Backup Flag</td>
<td>271</td>
</tr>
<tr>
<td>Set Auditid</td>
<td>273</td>
</tr>
<tr>
<td>Set Config Option</td>
<td>274</td>
</tr>
<tr>
<td>Statistics Directory Cache Information</td>
<td>276</td>
</tr>
<tr>
<td>Statistics iobyaggr Information</td>
<td>279</td>
</tr>
<tr>
<td>Statistics iobydasd Information</td>
<td>283</td>
</tr>
<tr>
<td>Statistics iocounts Information</td>
<td>288</td>
</tr>
<tr>
<td>Statistics Kernel Information</td>
<td>291</td>
</tr>
<tr>
<td>Statistics Locking Information</td>
<td>294</td>
</tr>
<tr>
<td>Statistics Log Cache Information</td>
<td>298</td>
</tr>
<tr>
<td>Statistics Metadata Cache Information</td>
<td>301</td>
</tr>
<tr>
<td>Statistics Storage Information</td>
<td>305</td>
</tr>
<tr>
<td>Statistics Transaction Cache Information</td>
<td>309</td>
</tr>
<tr>
<td>Statistics User Cache Information</td>
<td>313</td>
</tr>
<tr>
<td>Statistics Vnode Cache Information</td>
<td>317</td>
</tr>
<tr>
<td>Unquiesce Aggregate</td>
<td>321</td>
</tr>
</tbody>
</table>

### Part 3. Appendixes

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix A. Running the zFS pfctl APIs in 64-bit mode</td>
<td>327</td>
</tr>
</tbody>
</table>

Statistics iocounts information (64-bit mode) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ..
## Tables

1. Determining sysplex-awareness for zFS read/write file systems .......................... 54
2. DATASET report fields ............................................... 76
3. FILE report fields .................................................. 76
4. IOBYDASD report fields ........................................... 77
5. LFS report sections .................................................. 81
6. STKM report fields .................................................. 89
7. STOR report fields ................................................... 92
9. zFS man command examples ........................................ 125
10. Summary of zFS pscctl APIs ........................................ 209
11. Summary of zFS w_pioctls .......................................... 210
About this document

The purpose of this document is to provide complete and detailed guidance and reference information. This information is used by system administrators that work with the z/OS File System (zFS) component of the IBM® z/OS® Distributed File Service base element.

How this document is organized

This document is divided into parts, each part divided into chapters:

- **Part 1, “zFS administration guide,” on page 1** provides guidance information for the z/OS File System (zFS).
- **Part 2, “zFS administration reference,” on page 117** provides reference information about z/OS File System (zFS), which includes z/OS system commands, zFS commands, and zFS data sets.

Conventions used in this document

This document uses the following typographic conventions:

**Bold** Bold words or characters represent system elements that you must enter into the system literally, such as commands.

**Italic** Italicized words or characters represent values for variables that you must supply.

**Example Font** Examples and information displayed by the system are printed using an example font that is a constant width typeface.

[ ] Optional items found in format and syntax descriptions are enclosed in brackets.

{} A list from which you choose an item found in format and syntax descriptions are enclosed by braces.

| A vertical bar separates items in a list of choices.

< > Angle brackets enclose the name of a key on a keyboard.

... Horizontal ellipsis points indicated that you can repeat the preceding item one or more times.

\ A backslash is used as a continuation character when entering commands from the shell that exceed one line (255 characters). If the command exceeds one line, use the backslash character \ as the last nonblank character on the line to be continued, and continue the command on the next line.

**Note:** When you enter a command from this document that uses the backslash character (\), make sure you immediately press the Enter key and then continue with the rest of the command. In most cases, the backslash has been positioned for ease of readability.

# A pound sign is used to indicate a command is entered from the shell, specifically where root authority is needed (root refers to a user with a UID = 0).
z/OS information

This information explains how z/OS references information in other documents and on the web.

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

To find the complete z/OS library, including the z/OS Information Center, see z/OS Internet Library (http://www.ibm.com/systems/z/os/zos/bkserv/).

References to DFS information

Information about installing Distributed File Service components is found in z/OS Program Directory.

Information about z/OS File System messages and codes is found in z/OS Distributed File Service Messages and Codes.
How to send your comments to IBM

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

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

Include the following information:
• Your name and address.
• Your email address.
• Your telephone or fax number.
• The publication title and order number:
  z/OS V2R1.0 Distributed File Service zFS Administration
  SA23-6887-00
• The topic and page number that is related to your comment.
• The text of your comment.

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

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

If you have a technical problem

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

This part of the document discusses guidance information for the z/OS File System (zFS).

- Chapter 1, “z/OS File System overview,” on page 3
- Chapter 2, “Installing and configuring zFS,” on page 11
- Chapter 3, “Managing zFS processes,” on page 19
- Chapter 4, “Creating and managing zFS file systems using compatibility mode aggregates,” on page 21
- Chapter 5, “Using zFS in a shared file system environment,” on page 51
- Chapter 6, “Performing a backup of zFS,” on page 63
- Chapter 7, “Migrating data from HFS to zFS,” on page 67
- Chapter 8, “Performance and debugging,” on page 69
- Chapter 9, “Overview of the zFS audit identifier,” on page 113
Chapter 1. z/OS File System overview

The z/OS Distributed File Service z/OS File System (zFS) is a z/OS UNIX System Services (z/OS UNIX) file system that can be used in addition to the hierarchical file system (HFS). zFS file systems contain files and directories that can be accessed with z/OS UNIX application programming interfaces (APIs). These file systems can support access control lists (ACLs). zFS file systems can be mounted into the z/OS UNIX hierarchy along with other local (or remote) file system types (for example, HFS, TFS, AUTOMNT, and NFS). For more information about ACLs, see z/OS UNIX System Services Planning.

zFS can be used for all levels of the z/OS UNIX System Services hierarchy (including the root file system). Because zFS has higher performance characteristics than HFS and is the strategic file system, HFS might not be supported in any future releases, which will cause you to migrate the remaining HFS file systems to zFS.

zFS can run sysplex-aware for read/write mounted file systems and for read-only mounted file systems. For more information, see “Specifying zFS file systems as sysplex-aware” on page 15, “Terminology and concepts” on page 4, and Chapter 5, “Using zFS in a shared file system environment,” on page 51.

Beginning with z/OS V1R13, zFS has enhanced its sysplex-aware support. For many file operations, zFS can now directly access zFS read/write mounted file systems in a shared file system environment from zFS client systems. In z/OS V1R13 and later releases, when zFS runs in a shared file system environment, zFS always runs sysplex-aware on a file system basis (sysplex=filesys). See “zFS enhanced sysplex-aware support” on page 55 for more information.

zFS and HFS can both participate in a shared sysplex. However, only zFS supports security labels. Therefore, in a multilevel-secure environment, you must use zFS file systems instead of HFS file systems. See z/OS Planning for Multilevel Security and the Common Criteria for more information about multilevel security and migrating your HFS version root to a zFS version root with security labels.

Note:
1. Beginning with z/OS V2R1, zFS no longer supports multi-file system aggregates. If you have data stored in zFS multi-file system aggregates, copy that data from the zFS multi-file system aggregate file systems into zFS compatibility mode aggregates. Because zFS multi-file system aggregates cannot be mounted in z/OS V2R1, you must copy the data from any file systems contained in multi-file system aggregates into zFS compatibility mode file systems using a non-shared file system environment on a system running a release prior to z/OS V2R1.
2. Beginning with z/OS V2R1, zFS no longer supports clones. If you have read-only clone (.bak) file systems, you should delete them using the zfsadm delete command.

Features

zFS provides many features and benefits, which are described in the following sections:
Performance
zFS provides significant performance gains in many customer environments. zFS provides additional performance improvements when running sysplex-aware in a shared file system environment.

Restart
zFS reduces the exposure to loss of updates. zFS writes data blocks asynchronously and does not wait for a sync interval. zFS is a logging file system. It logs metadata updates. If a system failure occurs, zFS replays the log when it comes back up to ensure that the file system is consistent.

Aggregate movement
As a part of supporting read/write mounted file systems that are accessed as sysplex-aware, zFS automatically moves zFS ownership of a zFS file system to the system that has the most read/write activity. “Terminology and concepts” has an explanation of z/OS UNIX file system ownership and zFS file system ownership. Chapter 5, “Using zFS in a shared file system environment,” on page 51 contains details.

Terminology and concepts
To present all the benefits and details of zFS administration, the following concepts and terminology are introduced:

Attach
When a zFS file system is mounted, the data set is also attached. Attach means that zFS allocates and opens the data set. This attach occurs the first time a file system contained in the data set is mounted.

A zFS data set can also be attached (by issuing the `zfsadm attach` command) without mounting it. However, there are many restrictions in this case. For example, the zFS data set would not be available to z/OS UNIX applications because it was not mounted. In a shared file system environment, the zFS data set would be detached, not moved, if the system went down or zFS internally restarted. You might attach a zFS data set to explicitly grow it (`zfsadm grow`) or to determine the free space available (`zfsadm aggrinfo`). You can also delete a .bak file system (`zfsadm delete`) from an attached zFS data set. Because zFS has removed support for .bak (clone) file systems, you should delete any .bak file systems before mounting the primary file system. You must detach the zFS data set (`zfsadm detach`) before mounting it.

Catch-up mount
When a file system mount is successful on a system in a shared file system environment, z/OS UNIX automatically issues a corresponding local mount, which is called a catch-up mount, to every other system’s PFS for a zFS read/write mounted file system that is mounted RWSHARE or for a read-only mounted file system.

If the corresponding local mount is successful, z/OS UNIX does not function ship from that system to the z/OS UNIX owning system when that file system is accessed. Rather, the file request is sent directly to the local PFS. This is sometimes referred to as Client=N, as indicated by the output of the `D OMVS,F` operator command, or `df -v` shell command. If the corresponding local mount is unsuccessful (for instance, DASD is not accessible from that system), z/OS UNIX function ships requests to the z/OS UNIX owning system when that file system is accessed (message BPXF221I might be issued). This is sometimes referred to as Client=Y, as
indicated by the output of the D OMVS,F or df -v commands. For examples of the command output, see "Determining the file system owner" on page 56.

File system ownership
IBM defines a file system owner as the system that coordinates sysplex activity for a particular file system. In a shared file system environment, there is also the concept of file system ownership. The owner of a file system is the first system that processes the mount. This system always accesses the file system locally; that is, the system does not access the file system through a remote system. Other non-owning systems in the sysplex access the file system either locally or through the remote owning system, depending on the PFS and the mount mode.

The file system owner is the system to which file requests are forwarded when the file system is mounted non-sysplex aware. Having the appropriate owner is important for performance when the file system is mounted read/write and non-sysplex aware. The term z/OS UNIX file system owner refers to the owner of the zFS file system as z/OS UNIX recognizes it. This is typically the system where the file system is first mounted, but it can differ from the zFS file system owner (see zFS file system owner).

zFS file system owner
zFS has its own concept of file system ownership, called the zFS file system owner. This is also typically the system where the file system is first mounted in a sysplex-aware environment. File requests to sysplex-aware file systems are sent directly to the local zFS PFS, rather than being forwarded to the z/OS UNIX file system owner. This concept is shown in Figure 1 on page 6. The local zFS PFS forwards the request to the zFS file system owner, if necessary. The z/OS UNIX file system owner can be different from the zFS file system owner. (In reality, zFS owns aggregates. Generally, we simplify this to say zFS file system owner because zFS compatibility mode aggregates only have a single file system.)

z/OS UNIX file system owner
The term z/OS UNIX file system owner refers to the owner of the zFS file system as z/OS UNIX knows it. This is typically the system where the file system is first mounted.

For details about sysplex considerations and the shared file system environment, see "Determining the file system owner" on page 56 and Chapter 5, “Using zFS in a shared file system environment,” on page 51.
Function shipping
Function shipping means that a request is forwarded to the owning system and the response is returned to the requestor through XCF communications.

Local mount
A local mount means that z/OS UNIX issues a successful mount to the local PFS, which in this case is zFS. z/OS UNIX does this when either the file system is mounted sysplex-aware for that mode (read/write or read-only) or the system is the z/OS UNIX owner. When a file system is locally mounted on the system, z/OS UNIX does not function ship requests to the z/OS UNIX owning system. To determine if a system has a local mount, see “Determining the file system owner” on page 56.

Non-sysplex aware (sysplex-unaware)
A file system is non-sysplex aware (or sysplex-unaware) if the PFS (Physical File System) supporting that file system requires it to be accessed through the remote owning system from all other systems in a sysplex (allowing only one connection for update at a time) for a particular mode (read-only or read/write). The system that connects to the file system is called the file system owner. Other system's access is provided through XCF communication with the file system owner. For a non-sysplex aware zFS file system, file requests for read/write mounted file systems are function shipped to the owning system by z/OS UNIX. The owning system is the only system where the file system is locally mounted and the only system that does I/O to the file system. See zFS file system owner and z/OS UNIX file system owner.

Read-only file system
A file system that is mounted for read-only access is a read-only file system.

read/write file system
A file system that is mounted for read and write access is a read/write file system.

Shared file system environment
The shared file system environment refers to a sysplex that has a BPXPRMxx specification of SYSPLEX(YES).
Sysplex
The term *sysplex* as it applies to zFS, means a sysplex that supports the z/OS UNIX shared file system environment. That is, a sysplex that has a BPXPRMxx specification of SYSPLEX(YES).

Sysplex-aware
Pertains to a physical file system that handles file requests for mounted file systems locally instead of shipping function requests through z/OS UNIX.

Sysplex-aware PFS
A physical file system (PFS), for example zFS, is sysplex-aware or non-sysplex aware for a particular mount mode (read-only or read/write) in a shared file system environment. When it is sysplex-aware, the PFS is capable of handling a local mount on the system that is not the z/OS UNIX owning system. The PFS that is sysplex-aware can avoid z/OS UNIX function shipping for that mode. Both HFS and zFS file systems are always sysplex-aware for read-only mounts. HFS is always non-sysplex aware for read/write mounts and always results in z/OS UNIX function shipping from systems that are not the z/OS UNIX owning system. As of z/OS V1R13, zFS always runs sysplex-aware (sysplex=filesys) in a shared file system environment. Individual file systems can be non-sysplex aware or sysplex-aware, with the default being non-sysplex aware.

Sysplex-aware file system
A file system can be mounted sysplex-aware or non-sysplex aware. When a file system is mounted sysplex-aware, it means that the file system is locally mounted on every system (when the PFS is capable of handling a local mount on every system - that is, the PFS is running sysplex-aware) and therefore, file requests are handled by the local PFS. All read-only mounted file systems are always mounted sysplex-aware (see Figure 18 on page 52). HFS read/write mounted file systems are always mounted non-sysplex aware. This means that file requests from non z/OS UNIX owning systems are always function shipped by z/OS UNIX to the z/OS UNIX owning system where the file system is locally mounted and the I/O is actually done.

Beginning with z/OS V1R11, zFS read/write mounted file systems can be mounted sysplex-aware (see Figure 20 on page 53 and Figure 21 on page 54) when zFS is configured as sysplex-aware (zFS IOEFSPRM option sysplex=on or zFS IOEFSPRM option sysplex=filesys). Beginning with z/OS V1R13, zFS in a shared file system environment is always sysplex=filesys.

zFS aggregate
The data set that contains a zFS file system is called a zFS aggregate. A zFS aggregate is a Virtual Storage Access Method (VSAM) linear data set. After the zFS aggregate is defined and formatted, a zFS file system is created in the aggregate. In addition to the file system, a zFS aggregate contains a log file and a bitmap describing the free space. A zFS aggregate has a single read/write zFS file system and is sometimes called a compatibility mode aggregate. Compatibility mode aggregates are similar to HFS.

Restriction: zFS does not support the use of a striped VSAM linear data set as a zFS aggregate. If you attempt to mount a compatibility mode file system that had previously been formatted and is a striped VSAM linear data set, it will only mount as read-only. zFS does not support a zFS aggregate that has guaranteed space.
zFS file system

Refers to a hierarchical organization of files and directories that has a root directory and can be mounted into the z/OS UNIX hierarchy. zFS file systems are located on DASD.

zFS Physical File System (PFS)

Refers to the code that runs in the zFS address space. The zFS PFS can handle many users accessing many zFS file systems at the same time.

What's new or changed for zFS in z/OS V2R1

Beginning with z/OS V2R1, zFS no longer supports multi-file system aggregates and clones. As a result, the following zfsadm commands are no longer supported:

- `zfsadm clone`
- `zfsadm clonesys`
- `zfsadm create`
- `zfsadm lsquota`
- `zfsadm rename`
- `zfsadm setquota`

The following options are no longer supported on `zfsadm config`:

- `-fsgrow`
- `-user_cache_readahead`

The following options are no longer supported on `zfsadm configquery`:

- `-auto_attach`
- `-fsgrow`
- `-user_cache_readahead`

The following `pfsct1` subcommands are no longer supported:

- On the Aggregate command:
  - Create File System
- On the File System command:
  - Clone File System
  - Rename File System
  - Set File System Quota
- On the Config command:
  - Query auto_attach setting
  - Query fsgrow setting
  - Set fsgrow
  - Set user_cache_readahead

If you are using multi-file system aggregates or clones, you must stop using them. Be sure that you complete the migration actions described in z/OS Migration.

The zFS salvager program (ioeagslv) has been improved in z/OS V2R1:

- It can process larger zFS file systems by using storage above the 2 GB bar.
- It can complete its repair processing without needing to be run multiple times.
- All messages that it issues have message numbers.
- The verify option (-verifyonly) replays the log when necessary. This replay avoids reports of inconsistencies that occur when the log has not been replayed.

Quiesce processing for zFS file systems has been modified in z/OS V2R1. The zFS commands and zFS APIs used to quiesce and unquiesce zFS file systems are unchanged, but the way quiesce works internally and the way the quiesce status is displayed are modified.
In z/OS V2R1, the name “zSeries® File System” was changed to “z/OS File System”. The document z/OS Distributed File Service zSeries File System Administration was retitled to z/OS Distributed File Service zFS Administration.

Beginning with z/OS V2R1, zFS provides an optional, new format zFS aggregate, the version 1.5 aggregate. The current zFS aggregates are version 1.4 aggregates. The main purpose of the version 1.5 aggregate is to support a new directory format (extended (v5) directory) that will scale better when the directory contains many names (over 10,000). Since the format of a new directory is different in a version 1.5 aggregate, zFS provides toleration APAR OA39466 to cause a mount of a version 1.5 aggregate in an earlier release to fail. Earlier releases cannot access extended (v5) directories or version 1.5 aggregates. In order to control the transition to the new format directories, extended (v5) directories can only be created in version 1.5 aggregates. To create or change to a version 1.5 aggregate, you must explicitly request it. By default, aggregates created in z/OS V2R1 are version 1.4 aggregates. You should only create or change to a version 1.5 aggregate if you are sure you will not run releases prior to z/OS V2R1. Over time (possibly several releases), most zFS aggregates will be version 1.5 aggregates. IBM is likely to then change the default to version 1.5.

zFS toleration APAR OA39466 applies to z/OS V1R12 and V1R13.

zFS recommends that you should begin using the new zFS batch utility program IOEFSUTL. It contains all the function of the zFS format utility (IOEAGFMT) and the zFS salvage utility (IOEAGSLV). IOEFSUTL supports both version 1.5 aggregates and version 1.4 aggregates.

Beginning with z/OS V2R1, the batch utility ioeagfmt requires that the ZFS PFS be active.

New IOEPRMxx configuration options control what version an aggregate is formatted as by default (format_agrversion), whether a version 1.4 aggregate is changed to a version 1.5 aggregate on mount (change_agrversion_on_mount) and whether directories are converted to extended (v5) directories as they are accessed (converttov5).

A new MOUNT PARM controls whether a particular zFS aggregate's directories are converted to extended (v5) directories as they are accessed (CONVERTTOV5).

zFS has enhanced its support for the backup change activity flag in the VTOC (D1DSCHA in the Format 1/8). This flag indicates whether a backup of the file system is needed (that is, data has been modified in the file system since the last backup).

Beginning with z/OS V2R1, the default value for IOEPRMxx configuration options user_cache_size, meta_cache_size, and metaback_cache_size are now calculated based on the amount of real storage in the system.

Beginning with z/OS V2R1, the default will be to create zFS auditfids during aggregate formatting.

A new configuration variable was added to IOEFSPRM: user_running_hangdump.

To help alleviate the version 4 large directory performance problem before migrating to version 1.5 aggregates, zFS will allow the creation of new Large Fast Lookup Cache buffers above the bar (64 bit storage) that will be used to fully cache...
large directories. This is done with a new IOEPRMxx configuration option f1c. This option will only be valid in releases z/OS V1R13 and V2R1. It is available on z/OS V1R13 in APAR OA40530.
Chapter 2. Installing and configuring zFS

zFS is part of the Distributed File Service base element of z/OS. Before using the zFS support, you must install the z/OS release, the Distributed File Service, and the other base elements of z/OS using the appropriate release documentation.

Note: If you are only using the zFS support of the Distributed File Service (and not the SMB server support of the Distributed File Service), SMB does not need to be configured. For more information about SMB, see z/OS Distributed File Service SMB Administration.

To use the zFS support, you must configure the support on the system. Configuration includes the following administrative tasks:

- Define the zFS physical file system to z/OS UNIX
- Create or update the zFS parameter data set (IOEFSPRM); see “IOEFSPRM” on page 193.
- Define zFS aggregates and file systems
- Create mount points and mount zFS file systems
- Change owner/group and set permissions on file system root
- Optionally, add MOUNT statements in your BPXRPRMxx member to cause zFS file systems to be mounted at IPL.

zFS installation and configuration steps

To install, configure, and access zFS, you must perform the following administrative steps:

1. Install and perform post-installation of the Distributed File Service by following the applicable instructions in the z/OS Program Directory or in ServerPac: Installing Your Order. The following is a summary of the information that is contained in those documents:

   a. Ensure that the target and distribution libraries for the Distributed File Service are available.
   b. Run the prefix.SIOESAMP(IOEISMKD) job from UID 0 to create the symbolic links that are used by the Distributed File Service. This job reads the member prefix.SIOESAMP(IOEMKDIR) to delete and create the symbolic links.
   c. Ensure that the DDDEFS for the Distributed File Service are defined by running the prefix.SIOESAMP(IOEISDDD) job.
   d. Install the Load Library for the Distributed File Service. The Load Library (hlq.SIEALNKE) must be APF-authorized and must be in link list.
   e. Install the samples (hlq.SIOESAMP).
   f. Install the sample PROC for ZFS (hlq.SIOEPROC).
   g. Create a JCL PROC for the ZFS started task in SYS1.PROCLIB by copying the sample PROC from the previous step.

   The DDNAME IOEZPRM identifies the optional zFS configuration file. Although this DD statement is optional, it is recommended that it be included to identify the parameter data set to be used for ZFS. For now, it is suggested that this DD refer to a PDS with a member called IOEFSPRM that
has a single line that begins with an asterisk (*) in column 1. Subsequent modifications can be made to the IOEFSPRM member, see "IOEFSPRM" on page 193.

As the preferred alternative to the IOEZPRM DDNAME specification, delete the IOEZPRM DDNAME and use the IOEPRMxx PARMLIB member. In this case, the member has the name IOEPRMxx, where you specify xx in the PARMLIB member list. See "IOEFSPRM" on page 193 for more information.

To run zFS so that it is not under control of JES, see step 2. You might want to do this so that zFS does not interfere with shutting down JES.

h. Add the following RACF® commands:

```
ADDGROUP DFSGRP SUPGROUP(SYS1) OMVS(GID(2))
ADDOUSER DFS OMVS(HOME('/opt/dfslocal/home/dfscntl') UID(0))
DELTGRP(DFSGRP) AUTHORITY(USE)UACC(NONE)
RDEFINE STARTED DFS.** STDATA(USER(DFS))
RDEFINE STARTED ZFS.** STDATA(USER(DFS))
SETROPTS RACLST(STARTED)
SETROPTS RACLST(STARTED) REFRESH
```

The DFS user ID must have at least ALTER authority to all VSAM linear data set that contain zFS aggregates. A user ID other than DFS can be used to run the zFS started task if it is defined with the same RACF characteristics as shown for the DFS user ID. As an alternative to PERMIT ALTER authority to all VSAM linear data set that contain zFS aggregates, you can assign the zFS started task the TRUSTED attribute or you can assign the user ID of the zFS started task the OPERATIONS attribute. For details, see [z/OS Security Server RACF Security Administrator’s Guide](#).

2. Create a BPXPRMxx entry for zFS.

Add the following FILESYSTYPE statement to your BPXPRMxx:

```
FILESYSTYPE TYPE(ZFS) ENTRYPNT(IOEFSCM) ASNAME(ZFS)
```

Update your IEASYSxx parmlib member to contain the OMVS=(xx,yy) parameter for future IPLs.

If necessary, you can specify that zFS should not run under control of JES by specifying SUB=MSTR as in the following example:

```
FILESYSTYPE TYPE(ZFS) ENTRYPNT(IOEFSCM) ASNAME(ZFS, 'SUB=MSTR')
```

To use the IOEPRMxx parmlib members (mentioned in step 1g on page 11), specify the xx values in the FILESYSTYPE statement for zFS as in the following example:

```
FILESYSTYPE TYPE(ZFS) ENTRYPNT(IOEFSCM) ASNAME(ZFS, 'SUB=MSTR') PARM('PRM=(01,02,03)')
```

In this case, you must not have an IOEZPRM DD statement in your ZFS PROC. For more information about using IOEPRMxx, see "IOEFSPRM" on page 193.

3. Run the `dfs_cpfiles` program.

Running this program as described in the [z/OS Program Directory](#) is recommended even if you plan to only use zFS support. The only zFS configuration file is the zFS configuration file, which is not created by the `dfs_cpfiles` program. But, to complete the installation of the Distributed File Service, run the `dfs_cpfiles` program to create other files that are needed by the SMB server. This avoids problems if SMB is later activated.

To run the `dfs_cpfiles` program:

- Logon as root (UID 0) on the local z/OS system.
- From the z/OS UNIX shell, enter `/usr/lpp/dfs/global/scripts/dfs_cpfiles`.

4. (Optional) Create or update the zFS configuration options file (IOEPRMxx, also known as IOEFSPRM).
The zFS configuration options file is optional. There are two methods to specify the zFS configuration options file: use IOEPRMxx in the parmlib (preferred) or use a IOEZPRM DD statement in your ZFS PROC.

- As the preferred alternative to the IOEZPRM DD statement, the IOEFSPRM member can be specified as a true parmlib member. In this case, the member has the name IOEPRMxx, where xx is specified in the parmlib member list. You must omit the IOEZPRM DD statement in your ZFS PROC to use IOEPRMxx. The IOEPRMxx configuration options file can be specified with no options contained in it. Options are only required if you want to override the default zFS options. As mentioned in step 1g on page 11, it is recommended that you create an empty IOEPRMxx. The IOEPRMxx member should only contain one line that is a comment (an asterisk (*) in column 1). See “IOEFSPRM” on page 193 for more information.

- If you use the IOEZPRM DD statement, the PDS (organization PO) to which it points should have a record format of FB with a record length of 80. The block size can be any multiple of 80 that is appropriate for the device. A sample IOEFSPRM is provided in hlq.SIOESAMP.IOEFSPRM. IOEFSPRM is also known as IOEZS001. See “IOEFSPRM” on page 193 for a description of the IOEFSPRM options. Update the IOEZPRM DD statement in the ZFS PROC to contain the name of the IOEFSPRM member, as shown in the following example:

  IOEZPRM DD DSN=SYS4.PVT.PARMLIB(IOEFSPRM),DISP=SHR

  If you are running a sysplex, you must have different zFS configuration files for different systems. Chapter 5, “Using zFS in a shared file system environment,” on page 51 explains why different zFS configuration files are required. In this case, you should also specify a system qualifier in the data set name in the IOEZPRM DD, as shown in the following example:

  IOEZPRM DD DSN=SYS4.&SYSNAME..PARMLIB(IOEFSPRM),DISP=SHR

5. (Optional) Preallocate data sets for debugging.

   This step is optional because trace information is always available in the dump data set, and can be requested only by IBM Service. If needed, allocate the zFS trace output data set as a PDSE with RECFM=VB, LRECL=133 with a primary allocation of at least 50 cylinders and a secondary allocation of 30 cylinders. The name of this trace output data set should be specified in the trace_dsn option in the IOEFSPRM file. Next, allocate a debug settings data set as a PDS member with an LRECL=80. Add one comment line in the member (use a /* followed by */). Specify the name of this debug settings data set member in the debug_settings_dsn option of the IOEFSPRM file. Perform this process for each member of the sysplex.

6. Create a zFS (compatibility mode) file system.

   A zFS file system resides in a zFS aggregate. A zFS aggregate is a VSAM linear data set. See Chapter 4, “Creating and managing zFS file systems using compatibility mode aggregates,” on page 21 for details on creating zFS file systems.

   Beginning in z/OS V2R1, ioeagfmt fails if the zFS PFS is not active on the system.

7. Create a directory and mount the zFS file system on it.

   You can create a directory with the z/OS UNIX mkdir command or you can use an existing directory. The TSO/E MOUNT command or the /usr/sbin/mount REXX exec can be used to mount the zFS file system on the directory. See Chapter 4, “Creating and managing zFS file systems using compatibility mode aggregates,” on page 21 for details on mounting zFS file systems.
Note: Steps 6 on page 13 and 7 on page 13 can be repeated as many times as necessary for each permanently mounted zFS file system. Only step 6 on page 13 is needed for zFS automounted file systems (assuming that the automount file system has been set up.)

8. Add mount statements to BPXPRMxx members to mount the zFS file systems on the next IPL.

For example:

```
MOUNT FILESYSTEM('OMVS.PRIV.COMPAT.AGGR001') TYPE(ZFS) MOUNTPOINT('/etc/mountpt')
```

Applying required APARs for z/OS V2R1

In z/OS V2R1, in addition to the “zFS installation and configuration steps” on page 11, you must apply zFS coexistence function, using the following APAR procedure:

1. Install APAR OA39466 on all z/OS V1R12 and z/OS V1R13 systems. This APAR is a conditioning function for zFS on z/OS V1R12 and V1R13. Make APAR OA39466 active on all systems through a rolling IPL.

2. After APAR OA39466 is active on each z/OS V1R12 and V1R13 system, specify the sysplex=filesys configuration option in the IOEFSPRM file (if you do not already have sysplex=filesys specified on all your systems). Make this change active on all z/OS V1R12 systems through another rolling IPL. Because zFS R13 always runs as sysplex=filesys, this is now also required for zFS on z/OS V1R12 systems. (The default for sysplex in z/OS V1R12 is sysplex=off). For more information about specifying zFS configuration options, see “IOEFSPRM” on page 193.

Before you attempt to bring zFS V2R1 into the shared file system environment, APAR OA39466 must be installed and active and sysplex=filesys must be active on all z/OS V1R12 systems.

The sysplex configuration option cannot be specified dynamically through the zfsadm config command.

When sysplex=filesys, the output of the MODIFY ZFS,QUERY,LEVEL operator command indicates the status of the sysplex option when running in a shared file system environment. The following example shows the results from a z/OS V1R12 system displaying sysplex=filesys.

```
f zfs,query,level
IOEZ00639I zFS kernel: z/OS zSeries File System
Version 01.12.00 Service Level OAnnnnn - HZFS3B0.
sysplex(filesys,norwshare) interface(3)
```

If a problem occurs when zFS is running sysplex=filesys on a z/OS V1R12 system, you can perform the following steps:

- Remove the sysplex specification from each system or specify sysplex=off on each system (this is equivalent to the default)
- Perform a rolling IPL or restart zFS on each system.

Note that this procedure cannot be done after zFS on the z/OS V1R13 system has joined the sysplex. Also, if you try to start zFS on another z/OS V1R13 system after you have changed zFS to sysplex=off on the z/OS V1R12 system, zFS on z/OS V1R13 will not start. This happens because zFS on z/OS V1R13 requires all other systems be zFS sysplex=filesys.
Tip: Use the zFS migration check, ZOSMIGV1R13_ZFS_FILESYS, in the IBM Health Checker for z/OS to help determine if you are running zFS at the correct sysplex level. This check is automatically installed with APAR OA35465. See IBM Health Checker for z/OS: User’s Guide.

Specifying zFS file systems as sysplex-aware

This section helps you determine whether to make a zFS read/write file system be sysplex-aware.

If you are running your sysplex in a shared file system environment, where BPXPRMxx specifies SYSPLEX(YES), zFS V1R13 and z/OS V2R1 is always enabled to allow zFS read/write sysplex-aware file systems (zFS runs sysplex=filesys). You can individually choose which file systems are sysplex-aware for read/write and which ones are not. The default is that zFS read/write file systems will not be sysplex-aware. A newly mounted zFS read/write file system will be sysplex-aware if you specify the RWSHARE MOUNT PARM, as shown:

MOUNT FILESYSTEM('OMVS.PRV.COMAT.AGGR001') TYPE(ZFS) MOUNTPOINT('/etc/mountpt') PARM('RWSHARE')

As an alternative, you can specify sysplex_filesys_sharemode=rwshare in your IOEFSPRM. This changes the default so that each zFS read/write file system is mounted sysplex-aware (unless you explicitly specify the NORWSHARE MOUNT PARM).

Typically, if you make a zFS read/write file system sysplex-aware, you see a performance improvement in most shared file system environments when accessing the data from a system that is not the zFS owner. However, some servers cannot fully support zFS read/write file systems that are sysplex-aware.

• The z/OS Distributed File Service SMB server cannot export any zFS read/write file systems that are sysplex-aware.
• The Fast Response Cache Accelerator support of the IBM HTTP Server for z/OS V5.3 uses an API called register file interest (BPXIIOC using the locc#RegFileInt subcommand). This API cannot support zFS sysplex-aware read/write file systems, so therefore the Cache Accelerator support is not able to cache static Web pages contained in files in a zFS read/write sysplex-aware file system. Other servers that use this API can also be impacted. Generally, these are servers that cache files and must be aware of file updates from other sysplex members without having the server read the file or the file modification timestamp.
• The Policy Agent (Pagent) server, which is part of the z/OS Communications Server, cannot export any zFS read/write file systems that are sysplex-aware.

If you are using any of these servers, you should ensure that any zFS read/write file systems that are accessed by these servers are non-sysplex aware.

Note that there are some modifications to the way file system ownership works for zFS read/write sysplex-aware file systems. These modifications can cause some operational differences. For information about file system ownership, see Chapter 5, “Using zFS in a shared file system environment,” on page 51.

Using zFS read/write sysplex-aware file systems

When you run zFS V2R1 in a shared file system environment, the zFS PFS runs as sysplex-aware. However, by default, each zFS file system is mounted as
non-sysplex aware. zFS does allow zFS read/write file systems to run as
sysplex-aware. But, this must be explicitly requested on a file system basis, using
either the RWSHARE MOUNT PARM or the sysplex_filesys_sharemode=rwshare
configuration option.

Consider which zFS read/write file systems you might want to be sysplex-aware.
Good candidates are zFS read/write file systems that are accessed from multiple
systems or are mounted with AUTOMOVE and might be moved by z/OS UNIX
(as a result of a shutdown or IPL) to systems that do not necessarily do the most
accesses. Be aware that RWSHARE file systems use more virtual storage in the zFS
address space and you must be careful that you do not exhaust the zFS address
space storage. See the sample zFS query report “STOR” on page 89 for information
about monitoring storage usage in the zFS address space. Generally, the
system-specific file system (and /dev, /etc, /tmp, /var) should be mounted as
NORWSHARE and UNMOUNT because they typically are accessed only from the
owning system.

zFS read-only mounted file systems are not affected by this support. However, if
you remount a read-only file system to read/write (using the chmount command
or the TSO/E UNMOUNT REMOUNT command), this is treated like a primary
mount on the current z/OS UNIX owning system. In this case, MOUNT
parameters (such as RWSHARE or NORWSHARE) or MOUNT defaults (such as
the current sysplex_filesys_sharemode setting on that system) take affect when it is
mounted read/write. When you remount back to read-only, those mount options
are irrelevant again.

Note: These MOUNT parameters and MOUNT defaults do not take effect when a
remount to the same mode is run.

The sysplex_filesys_sharemode option on a system specifies if a zFS read/write file
system will be mounted as sysplex-aware when a MOUNT is issued on that
system without specifying either NORWSHARE or RWSHARE in the MOUNT
PARM. The default value for sysplex_filesys_sharemode is norwshare. This means
that a MOUNT for a zFS read/write file system that does not have NORWSHARE
or RWSHARE specified in the MOUNT PARM will result in the file system being
non-sysplex aware. If you want zFS read/write mounts to be sysplex-aware, then
specify sysplex_filesys_sharemode=rwshare. This option can be specified in the
IOEFSPRM configuration options file and takes effect on the next IPL (or restart of
zFS). It can also be specified dynamically with the zfsadm config
-sysplex_filesys_sharemode command. Typically, you should specify the same
sysplex_filesys_sharemode value on all your systems. Otherwise, z/OS UNIX file
system ownership movement might change the sysplex-awareness of a file system
that does not have NORWSHARE or RWSHARE specified in the MOUNT PARM.

Note: If any zFS read/write file systems were previously mounted as
NORWSHARE, they will usually remain non-sysplex aware until they are
un-mounted and then mounted back on the RWSHARE system. However, there are
situations when the sysplex awareness may change. See “Changing zFS attributes
on a mounted zFS compatibility mode file system” on page 38 for more
information.

Your sysplex root file system should be read-only. However, if your sysplex root
file system is normally read/write, you should make it sysplex-aware. You cannot
unmount the sysplex root file system so you need an alternative method. One
method is to remount your sysplex root to read-only, move z/OS UNIX ownership
of the file system, if necessary, to a system that has
sysplex_filesys_sharemode=rwshare, and then remount the sysplex root back to read/write. You may want to update your ROOT statement in BPXPRMxx to add PARM('RWSHARE') to ensure that you do not lose the sysplex-aware attribute if the ROOT is mounted again. Note that, in this case, you may see a USS_PARMLIB health check message indicating that your BPXPRMxx ROOT PARM does not match your current sysplex root PARM. This is expected and is normal.

**Changing the sysplex-awareness of a mounted zFS read/write file system**

In a shared file system environment, after a zFS read/write file system is mounted it is either sysplex-aware or non-sysplex aware. You can determine the sysplex-awareness of a mounted zFS read/write file system by using the `zfsadm aggrinfo -long` command. If it displays sysplex-aware, then it is sysplex-aware. If it is blank, then it is non-sysplex aware.

Alternatively, you can also issue the `f zfs,query,file` console command. As Figure 35 on page 76 shows, an “S” indicates that the zFS read/write file system is mounted sysplex aware. Because you do not have to be running in the shell, this command can be useful if a file system is under recovery or having other problems.

To change the sysplex-awareness of a mounted zFS read/write file system, you need to perform the following steps:

- Unmount the file system
- Specify the desired MOUNT PARM (RWSHARE to make it sysplex-aware; NORWSHARE to make it non-sysplex aware)
- Mount the file system again.

If you want to change the sysplex-awareness and you have not specified either the RWSHARE or NORWSHARE MOUNT PARM, you can change the sysplex-awareness with remount. To do so:

- Remount the file system to read-only
- Move z/OS UNIX ownership of the file system (if necessary) to a system that has sysplex_filesys_sharemode specified to the sharemode that you want (rwshare or norwshare)
- Remount the file system back to read/write.
Chapter 3. Managing zFS processes

This section describes the zFS address space. It also discusses starting and stopping zFS, and other activities required to manage zFS.

Starting zFS

zFS runs as a z/OS UNIX colony address space. There must be an entry in a BPXPRMxx parmlib member for ZFS and the ZFS PROC must be available. zFS is started by z/OS UNIX, based on the FILESYSTYPE statement for ZFS in the BPXPRMxx parmlib member. zFS can be started at IPL if the BPXPRMxx parmlib member is in the IEASYSxx parmlib member's OMVS=(xx,yy) list. It can also be started later by using the SETOMVS RESET=(xx) operator command.

Stopping zFS

In general, do not stop zFS. Stopping zFS is disruptive to applications that are using ZFS file systems. zFS stops automatically when you shut down z/OS UNIX. To shut down an LPAR or to re-IPL an LPAR, use the MODIFY OMVS,SHUTDOWN operator command to shut down z/OS UNIX. This action synchronizes data to the file systems and unmounts or moves ownership in a shared file system environment. A planned system shutdown must include the unmount or move of all owned file systems and the shut down of zFS. The MODIFY OMVS,SHUTDOWN command unmounts and moves the owned file systems and shuts down zFS. For shutdown procedures using OMVS,SHUTDOWN, see the topic on Planned shutdowns using OMVS,SHUTDOWN in z/OS UNIX System Services Planning.

zFS can be stopped using the MODIFY OMVS,STOPPFS=ZFS operator command. Automatic ownership movement can occur for both the z/OS UNIX owner and the zFS owner. See z/OS UNIX System Services Planning for information about the various automove settings for z/OS UNIX file system ownership. zFS aggregate ownership will move unless the file system is unmounted by z/OS UNIX. zFS file systems that become unmounted will need to be mounted again after zFS is restarted.

When zFS is stopped, you receive the following message (after replying Y to message BPXI078D):

nn BPXF032D FILESYSTYPE ZFS TERMINATED. REPLY ‘R’ WHEN READY TO RESTART. REPLY ‘I’ TO IGNORE.

When an LPAR is shut down without the orderly shutdown of zFS, it is likely that recovery actions (automatic recovery on the next mount; if the mount fails, it might be necessary to manually run salvager) will be necessary to bring zFS aggregates back to a consistent state. In addition, some file activity can be lost.

To restart zFS, reply r to message nn. (For example, r 1r). If you want zFS to remain stopped, you can reply i to remove the prompt. In this case, zFS can be redefined at a later time using the SETOMVS RESET=(xx) operator command.

However, this can result in zFS file systems becoming NOT ACTIVE. An unmount and remount is required to activate a file system that is NOT ACTIVE. If you plan to restart zFS, you should reply r to the message.
Note: Stopping zFS can have shared file system (sysplex) implications. See Chapter 5, "Using zFS in a shared file system environment," on page 51 for information about shared file systems.

If the zFS colony address space has an internal failure, it typically does not terminate. It might disable an aggregate (see “Diagnosing disabled aggregates” on page 110). If it is a case where it does terminate, normally the zFS colony address space will restart automatically. Otherwise, message BPXF032D (the same message you receive when the MODIFY OMVS,STOPPFS=ZFS operator command is used) is issued and a reply is requested.

On z/OS V1R13 and later systems, if an internal problem occurs, zFS attempts an internal restart. It internally remounts any zFS file systems that were locally mounted, without requiring any support from z/OS UNIX. The zFS ownership for aggregates that are owned on the system that is internally restarted might be moved (by zFS for sysplex-aware file systems) to another system. For more information, refer to Step 10 on page 109.

If zFS is running on a z/OS V1R12 or earlier system and terminates abnormally, automatic ownership movement occurs for both the z/OS UNIX owner and the zFS owner, if possible. See z/OS UNIX System Services Planning for information about z/OS UNIX file system ownership for the various automove settings. zFS aggregate ownership moves unless the file system is unmounted by z/OS UNIX. Applications with an open file on these file systems receive I/O errors until the file is closed. After zFS is restarted, the operator must remount any file systems that were locally mounted (that is, file systems that were owned by that system and were not moved). This can be done by using the MODIFY BPXOINIT,FILESYS=REINIT operator command. This causes a remount for each file system that was mounted through a BPXPRMxx parmlib statement.

Determining file system status

To determine if zFS is currently active, use the following steps:
1. If the BPXF032D message is outstanding, zFS is not active.
2. If the operator command D A,ZFS says zFS is not found, zFS is not active.
3. If the operator command D A,ZFS gives the zFS address space information, zFS is active.

For information about how to identify the file system owner, see "Determining the file system owner" on page 56. To display zFS internal restart information, issue the MODIFY ZFS,QUERY,STATUS operator command.

Beginning in z/OS V1R11, you can issue D OMVS,P to display the state of the PFS, including the start or exit timestamp. Message BPXO068I returns the PFS in one of the following possible states:

A  Active; the timestamp is the start time of the PFS.
I  Inactive. When the PFS is inactive with no timestamp, the PFS address space has not yet started. When the PFS is inactive with timestamp, the PFS has stop at that time.
S  Stopped; it is waiting for a reply of R to restart or I to terminate the PFS.
U  Unavailable.

For more information, see z/OS MVS System Messages, Vol 3 (ASB-BPX)
Chapter 4. Creating and managing zFS file systems using compatibility mode aggregates

This section discusses creating compatibility mode aggregates and file systems.

Creating a compatibility mode aggregate

A zFS file system is created in a zFS aggregate (which is a VSAM linear data set). When using compatibility mode aggregates, the aggregate and the file system are created at the same time. For simplicity, we refer to a file system in a compatibility mode aggregate as a compatibility mode file system. A compatibility mode file system is created using the `ioeagfmt` utility (which is described in "ioeagfmt" on page 125).

Creating a compatibility mode aggregate is a two-step process.

1. First, create a VSAM linear data set using IDCAMS. You must consider what the maximum size of the aggregate will be; if it will eventually exceed 4 G, you must define the aggregate as extended format, extended addressability. The VSAM linear data set must have a secondary allocation size specified, if you want to use dynamic grow. See “Dynamically growing a compatibility mode aggregate” on page 28 for more information.
   
   Tip: Consider defining the aggregate with a secondary extent because dynamic grow (aggrgrow) is the default.

2. Then format the VSAM linear data set as a compatibility mode aggregate and create a file system in the aggregate using `ioeagfmt` (see "ioeagfmt" on page 125). When using `ioeagfmt`, the user must meet one of the following authorization requirements:
   - Have ALTER authority to the VSAM linear data set.
   - Be UID 0.
   - Have read authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

You can also create a compatibility mode aggregate using the ISHELL, or the `zfsadm define` and `zfsadm format` commands. For more information about ISHELL, see "z/OS UNIX System Services User’s Guide". For more information about `zfsadm`, see "z/OS UNIX System Services Planning". For more information about the `zfsadm define` command, see "zfsadm define" on page 168. For more information about the `zfsadm format` command, see "zfsadm format" on page 178.

The VSAM linear data set, the aggregate, and the file system all have the same name and that name is equal to the VSAM linear data set cluster name. The zFS file system is then mounted into the z/OS UNIX hierarchy.

Rule: The Control Interval (CI) size of a VSAM linear data set that is formatted as a zFS aggregate must be 4 K. This is the default for IDCAMS. As such, it is not specified in Figure 2 on page 22, which shows an example of a job that creates a compatibility mode file system.
The -compat parameter in the CREATE step tells ioeagfmt to create a compatibility mode file system. As of z/OS V2R1, the -compat parameter is optional, and zFS always formats a compatibility mode file system. The result of this job is a VSAM linear data set that is formatted as a zFS aggregate and contains one zFS file system. The zFS file system has the same name as the zFS aggregate (and the VSAM linear data set). The size of the zFS file system (that is, its available free space) is based on the size of the aggregate.

The ioefsutl format utility can also be used to format a compatibility mode file system and has options similar to ioeagfmt. The -compat option is not needed or allowed. The ioefsutl format utility only formats compatibility mode aggregates. You are encouraged to use the ioefsutl format utility rather than the ioeagfmt utility.

The default for the size of the aggregate is the number of 8-KB blocks that fits in the primary allocation. You can specify a -size option giving the number of 8-KB blocks for the aggregate.

- If you specify a number that is less than (or equal to) the number of blocks that fits into the primary allocation, the primary allocation size is used.
If you specify a number that is larger than the number of 8-KB blocks that fits into the primary allocation, the VSAM linear data set is extended to the size specified if the total size will fit in the primary allocation and a single extension. A secondary extension cannot be used; instead, see “Growing a compatibility mode aggregate” on page 27. The single extension must be no larger than a single volume. This occurs during its initial formatting. Sufficient space must be available on the volume. Multiple volumes can be specified on the DEFINE of the VSAM linear data set. The multiple volumes are used during extension of the data set at a later time. If you want to create a multi-volume data set initially that is larger than two volumes, see “Creating a multi-volume compatibility mode aggregate” on page 29. DFSMS decides when to allocate on these volumes during extension. Any VSAM linear data set greater than 4 GB can be specified by using the extended format and extended addressability capability in the data class of the data set. See z/OS DFSMS Using Data Sets for information about VSAM data sets greater than 4 GB in size.

Restriction: zFS does not support the use of a striped VSAM linear data set as a zFS aggregate. If you attempt to mount a compatibility mode file system that was previously formatted and is a striped VSAM linear data set, it is mounted as read-only.

There are several other options to use when creating a compatibility mode file system that set the owner, group, and the permissions of the root directory.

- The -owner option specifies the owner of the root directory.
- The -group option specifies the group of the root directory.
- The -perms option specifies the permissions on the root directory.

Now, you can mount the zFS file system into the z/OS UNIX hierarchy with the TSO/E MOUNT command. For example, the following command mounts the compatibility mode file system that was created.

```
MOUNT FILESYSTEM('OMVS.PRV.COMPAT.AGGR001') TYPE(ZFS) MODE(RDWR) MOUNTPOINT('/usr/mountpt1')
```

Alternatively, as the following example shows, you can use the z/OS UNIX `mount` shell command to mount the compatibility mode file system that was created.

```
/usr/sbin/mount -t ZFS -f OMVS.PRV.COMPAT.AGGR001 /usr/mountpt1
```

These examples assume that the directory /usr/mountpt1 exists and is available to become a mount point. For more information about mount points, see z/OS UNIX System Services Planning.

When a zFS compatibility mode aggregate created before z/OS V1R7 is first mounted read/write on a later release, the on-disk format is modified from a version 1.3 aggregate to a version 1.4 aggregate. This action allows the performance of mount to improve (especially for zFS file systems with many files and directories). During the automatic conversion, you see messages such as:

```
IOEZ00500I Converting PLEX.JMS.AGGR007.LDS0007 for fast mount processing
IOEZ00618I Converting filesystem PLEX.JMS.AGGR007.LDS0007 to allow for fast mount
```
Using version 1.5 aggregates and extended (v5) directories

**Caution:** Do not use zFS version 1.5 aggregates until you have finished migrating to z/OS V2R1. Version 1.5 aggregates are not supported on previous releases. All systems in a sysplex must be at the V2R1 level before implementing any version 1.5 aggregates on any system in the sysplex.

Beginning in z/OS V2R1, zFS supports a new version aggregate, the *version 1.5 aggregate*. The current aggregates are version 1.4 aggregates. Version 1.5 aggregates support extended (v5) directories. Extended (v5) directories provide the following benefits:

- They can support larger directories with performance.
- They store names more efficiently than v4 directories.
- When names are removed from extended (v5) directories, the space is reclaimed, when possible, unlike v4 directories where space is not reclaimed until the directory is removed.

Version 1.5 aggregates have a larger architected maximum size than version 1.4 aggregates (approximately 16 TB versus approximately 4 TB). Also, extended (v5) directories can support more subdirectories than v4 directories (4G-1 versus 64K-1).

Because version 1.5 aggregates will benefit all z/OS V2R1 environments, you are encouraged to use this function after all of your systems have been migrated to z/OS V2R1. Version 1.5 aggregates can contain both extended (v5) directories and v4 directories and either can be a subdirectory of the other, while version 1.4 aggregates cannot contain extended (v5) directories. Version 1.5 aggregates can be mounted on directories that are contained in version 1.4 aggregates, and the reverse is also allowed.

**Creating a version 1.5 aggregate**

A version 1.5 aggregate can be created using one of the following methods:

- By formatting a VSAM linear data set as a version 5 using the zFS `ioefsutl format` batch utility
- Using the zFS `ioeagfmt` batch utility
- Via the Format Aggregate API
- Using the `zfsadm format` command

Version 1.5 aggregates are not formatted by default. They must be explicitly requested with the `-version5` option.

You can change the default version that is formatted by setting the IOEFSprm variable `format_aggrversion` configuration option to 5. The `format_aggrversion` value from the zFS PFS is used when any formatting method is used without the `-version4` or `-version5` parameters.

The zFS format utilities `ioeagfmt` and `ioefsutl format` both request the value of the `format_aggrversion` configuration option from the zFS kernel when determining the default aggregate version for the format. If the zFS PFS is down, both utilities will simply fail. Formatting of a version5 aggregate is not allowed when a z/OS V1R12 or z/OS V1R13 system is in a shared file system environment when using the batch utility `ioeagfmt`, the `zfsadm format` command or the Format Aggregate API.
Following is an example of a job to create and format a version 1.5 aggregate:

```plaintext
//USERIDA JOB ,'Compatibility Mode',
//CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//AMSDUMP DD SYSOUT=H
//DASD0 DD DISP=OLD,UNIT=3390,VOL=SER=PRV000
//SYSIN DD *
   DEFINE CLUSTER (NAME(OMVS.PRV.COMPAT.AGGR001) -
                   VOLUMES(PRV000) -
                   LINEAR CYL(25 10) SHAREOPTIONS(3))
/*
//CREATE EXEC PGM=IOEFSUTL,REGION=OM,
//PARM=('format -aggregate OMVS.PRV.COMPAT.AGGR001 -version5')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
/*
```

The `zfsadm format` command can also be used to format a version 1.5 aggregate.
For example:

```
# zfsadm define -aggr OMVS.PRV.ZFS.AGGR005.LDS0005 -volumes PRV000 -cyl 10 5
IOE200248I VSAM linear dataset OMVS.PRV.ZFS.AGGR005.LDS0005 successfully created.
# zfsadm format -aggr OMVS.PRV.ZFS.AGGR005.LDS0005 -version5
IOE200077I HFS-compatibility aggregate OMVS.PRV.ZFS.AGGR005.LDS0005 has
been successfully created
```

### Converting an existing aggregate to version 1.5

An existing version 1.4 aggregate can be changed to a version 1.5 aggregate and, optionally, existing directories contained in the aggregate can be converted to extended (v5) directories. Use any one of the following methods to change an aggregate to version 1.5.

- Explicitly, for a mounted aggregate using the `zfsadm convert -aggrversion` command, or
- Automatically, on mount when the `change_aggrversion_on_mount` configuration option is on (set in IOEPRMxx or using the `zfsadm config` command), or
- Automatically, on mount when the `converttov5` configuration option is on (set in IOEPRMxx or using the `zfsadm config` command), or
- Automatically, on mount when the `CONVERTTOV5 MOUNT PARM` is specified, or
- Offline, using the IOEFSUTL `converttov5` batch utility with the `-aggrversion_only` option.

**Note:** The `CONVERTTOV5` option and MOUNT PARM will also cause accessed directories to be converted to extended (v5) directories after the aggregate is converted to version 1.5.
An aggregate is not automatically changed if the NOCONVERTTOV5 MOUNT PARM is specified. An aggregate is not explicitly or automatically changed if there are earlier release systems (prior to z/OS V2R1) in the shared file system environment.

Following is an example of the `zfsadm convert` command to change a version 1.4 aggregate to a version 1.5 aggregate without converting any directories to extended (v5) directories:

```
# zfsadm convert -aggrversion OMVS.PRV.ZFS.AGGR005.LDS0005
IOEZ00810I Successfully changed aggregate OMVS.PRV.ZFS.AGGR005.LDS0005 to version 1.5
```

## Converting an existing v4 directory to an extended (v5) directory

Once an aggregate is a version 1.5 aggregate, new directories that are created in it will be extended (v5) directories. Existing directories can be converted to extended (v5) directories:

- Explicitly, one at a time, for a mounted aggregate using the `zfsadm convert -path` command, or
- Automatically, as they are accessed, for a mounted aggregate when the aggregate has the `converttov5` attribute, or
- Offline, converting all directories using the `ioefsutil converttov5` batch utility.

Existing directories in a version 1.5 aggregate are not automatically converted if the NOCONVERTTOV5 MOUNT PARM is specified. Explicit and offline directory conversion will change the aggregate from version 1.4 to 1.5, if necessary.

Following is an example of the `zfsadm convert` command to convert a v4 directory to an extended (v5) directory:

```
# zfsadm convert -path /home/suimgkp/zfsmnt5
IOEZ00791I Successfully converted directory /home/suimgkp/zfsmnt5 to version 5 format.
```

Converting a directory from version 4 to an extended (v5) directory requires both versions of the directory to exist on disk at the same time, temporarily. If the aggregate becomes full during the allocation of the new directory a dynamic grow will be attempted. See ["Dynamically growing a compatibility mode aggregate" on page 28](#) for information about controlling dynamic growth of an aggregate. If there is not enough space to complete the conversion, the new directory will be deleted and the conversion operation will fail.

When the conversion is completed, the old directory is deleted. The size of the resulting new directory will vary based on the actual directory contents. In some cases it may require more space than the original directory. In other cases it may require less space.

If a system outage occurs during a directory conversion, the directory will be made consistent during log recovery processing. That is, either the old directory will exist or the new directory will exist, but both will not exist.
Guidelines for v4 to v5 conversion

Extended (v5) directories have better performance than v4 directories of the same size. For optimal performance after all systems at your site have been migrated to z/OS V2R1, all of the directories should be converted from v4 to v5 even though support will continue to be provided for v4 directories. To convert selected file systems or directories, you can use automatic methods (such as specifying the MOUNT parameters or by using the offline conversion utility). You can also convert them explicitly with the `zfsadm convert` command.

If your installation exports zFS file systems to NFS or SMB, it is recommended that the `zfsadm convert` command not be used for conversions for directories that are exported by these servers. In rare cases, remote applications can get unexpected errors if a directory being manually converted is simultaneously being accessed by NFS or SMB users. Use one of the other methods for the conversion, such as offline conversion or the CONVERTTOV5 MOUNT parameter, for these file systems. These methods will ensure that each individual directory is completely converted before it can be exported.

If you are not planning to convert all file systems to v5, then it is best to at least do the most active file systems or the file systems with large directories. A directory will get a nontrivial benefit by conversion to v5 if it has 10000 entries or more (a length of approximately 800 K or more). You can determine the most active file systems by issuing `MODIFY ZFS,QUERY,FILESETS` or by using the `wjsfsmon` tool. The number of entries in a directory can be determined by issuing the command `df -t`. The approximate rate of conversion for the directories is between 3500 (for a z9® machine) and 10000 (for a zEC12 machine) directory entries per second, depending on your processor.

After you decide that a file system is going to be converted to v5, you need to decide what conversion method to use. If the file system can be unmounted, the `ioefsutl converttov5` batch utility or MOUNT parameters can be used. If it cannot be unmounted and it is not exported by NFS or SMB servers, use the `zfsadm convert` command. If it is exported by NFS or SMB servers, add the `converttov5` attribute to the mounted aggregate. See “Changing zFS attributes on a mounted zFS compatibility mode file system” on page 38 for instructions about how to add the `converttov5` attribute to the mounted file system.

Migrating data to version 1.5 aggregates

Data can be migrated from HFS file systems into a version 1.5 aggregate in much the same manner as it would be migrated into a version 1.4 aggregate. You can also copy data from a version 1.4 aggregate to a version 1.5 aggregate with the `z/OS UNIX shell command pax`. For more information, see Chapter 7, “Migrating data from HFS to zFS,” on page 67.

Growing a compatibility mode aggregate

If a compatibility mode aggregate becomes full, the administrator can grow the aggregate (that is, cause an additional allocation to occur and format it to be part of the aggregate). This is accomplished with the `zfsadm grow` command. There must be space available on the volume to extend the aggregate’s VSAM linear data set. The size that is specified on the `zfsadm grow` command must be larger than the current size of the aggregate.
For example, suppose a two cylinder (primary allocation, 3390) aggregate has a total of 180 8 KB blocks and a (potential) secondary allocation of one cylinder. 180 8-KB blocks is 1440 KB. A `zfsadm aggrinfo` command for this aggregate might show 1440 KB. When you issue the `zfsadm grow` command with a larger size, the file system becomes larger because DFSMS is called to allocate the additional DASD space.

```
zfsadm aggrinfo omvs.prv.aggr003.lds0003
OMVS.PRIV.AGRGRO03.LDS0003 (R/W COMP): 1279 K free out of total 1440
```

```
zfsadm grow omvs.orv.aggr003.lds0003 -size 1440
IOEZ00173I Aggregate OMVS.PRIV.AGRGRO03.LDS0003 successfully grown
OMVS.PRIV.AGRGRO03.LDS0003 (R/W COMP): 1279 K free out of total 1440
```

**Figure 4. Example zfsadm grow command specifying the same size as the current size**

In **Figure 4**, notice that the `zfsadm grow` command indicates success, but the aggregate was not made any larger because the size specified on the command was the same as the existing size.

```
zfsadm grow omvs.prv.aggr003.lds0003 -size 1441
IOEZ00173I Aggregate OMVS.PRIV.AGRGRO03.LDS0003 successfully grown
OMVS.PRIV.AGRGRO03.LDS0003 (R/W COMP): 1999 K free out of total 2160
```

**Figure 5. Example results from zfsadm grow command**

As shown in **Figure 5**, the aggregate now has a total size of 2160 KB. You can specify 0 for the size to get a secondary allocation size extension. The file system free space has also been increased based on the new aggregate size. Aggregates cannot be made smaller without copying the data to a new, smaller aggregate.

### Dynamically growing a compatibility mode aggregate

An aggregate can be dynamically grown if it becomes full. The aggregate (that is, the VSAM linear data set) must have secondary allocation specified when it is defined and space must be available on the volume. The number of extensions that are allowed is based on DFSMS VSAM rules (see [*z/OS DFSMS Using Data Sets*](https://www.ibm.com/support/docview.wss?rs=131&id=doc153770)). The aggregate is extended when an operation cannot complete because the aggregate is full. If the extension is successful, the operation is again transparently driven to the application.

An administrator can restrict aggregates from growing dynamically, either on an individual aggregate basis or globally. To restrict dynamic growing of a specific aggregate, use the NOAGGGRGROW parameter on the MOUNT command (see **“MOUNT” on page 146**). To globally restrict dynamic growing of all aggregates, specify the `aggrgrow=off` option of the IOEFSPRM configurations option file (see **“IOEFSPRM” on page 193**).

During the extension, a portion of the extension is formatted. Applications that cause new blocks to be allocated or that are reading a file that is being extended will wait. Other applications will not wait. Applications that must wait, will wait for the extension and the (portion) format. Look for HI-A-RBA, the size of the data set in bytes, and HI-U-RBA, how much of it is formatted in bytes. If the aggregate
has previously been extended but not fully formatted (that is, the HI-U-RBA (or hi-used-RBA) is less than the HI-A-RBA (or hi-allocated-RBA)), zFS will format another portion of the existing extension to make more space available. You can determine the HI-U-RBA and HI-A-RBA by using the IDCAMS LISTCAT ALL utility against the zFS aggregate and looking for HI-U-RBA and HI-A-RBA in the job output. Dividing HI-A-RBA or HI-U-RBA by 8192 will convert them to the number of 8K blocks.

Each time zFS formats a portion of the extension or each time zFS dynamically grows the aggregate and formats a portion of the extension, zFS issues message IOEZ00312I, and then issues one of the following messages:

- IOEZ00309I, when successful
- IOEZ00308E, when unsuccessful

When a dynamic extension fails (for example, because of insufficient space), zFS sets an internal indicator to avoid attempting another dynamic extension. This indicator can be reset by a successful explicit grow (for example, by using the zfsadm grow command) or by an unmount and mount of the file system.

Creating a multi-volume compatibility mode aggregate

Before you can create a large zFS aggregate (for example, ten full volumes), you must have the following prerequisites:

- Ten empty volumes
- A DFSMS DATACLASS that provides extended addressability (because the total size is greater than 4 GB)
- A JOB that defines and formats the aggregate.

Assuming that:

- Each volume is a 3390 with 3338 cylinders, and 3336 of those cylinders are free,
- There are 15 tracks per cylinder,
- And that you can get six 8-KB blocks per track (15 x 6 = 90 8 KB blocks per cylinder),

you should get 90 x 3336 = 300,240 8-KB blocks per volume and 10 x 300,240 = 3,002,400 8-KB blocks in the aggregate. [Figure 6 on page 30] is an example job that defines the VSAM linear data set in the first step and formats it as a zFS aggregate in the second step. The FORMAT step formats the primary allocation (3336 cylinders) and then extends the data set by the -grow amount (300,240 8-KB blocks) ten times (one extend for each full volume) until it reaches the total -size amount (3,002,400 8 KB blocks).

The example in [Figure 6 on page 30] causes the 10 full volumes to be allocated and formatted by using the -size and the-grow options on the IOEAGFMT step so that the result is a 10-volume (empty) file system. The -grow option is needed in order to allow the specification of a grow increment size that is less than the size of a volume.
As another example, you could define a VSAM linear data set as before with 10 volumes but with a secondary allocation size of 3336 cylinders, as shown in Figure 7. Then you could format only the first volume by leaving out the -size and the -grow and let zFS dynamic secondary allocation allocate and format the additional volumes (up to 9 more) as needed. The IOEPRMxx aggrgrow configuration option must be on.

```
USERIDA JOB ,'Multi-Volume',
CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
DEFINE EXEC PGM=IDCAMS
SYSPRINT DD SYSOUT=H
SYSSDUMP DD SYSOUT=H
AMSDUMP DD SYSOUT=H
SYSSIN DD *
DEFINE CLUSTER (NAME(OMVS.VOL10.COMPAT.AGGR001) -
 VOLUMES(PRV000 PRV001 PRV002 PRV003 PRV004 -
 PRV005 PRV006 PRV007 PRV008 PRV009) -
 DATACLASS(EXTATTR) -
 LINEAR CYL(3336) SHAREOPTIONS(3))
/*
//FORMAT EXEC PGM=IOEAGFMT,REGION=OM,
// PARM=('aggregate OMVS.VOL10.COMPAT.AGGR001 -compat -size 3002400 -gX
//row 300240')
//SYSPRINT DD SYSOUT=H
//STOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSSDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
/*@*
```

Figure 6. Example job to create a multi-volume compatibility mode aggregate

As another example, you could define a VSAM linear data set as before with 10 volumes but with a secondary allocation size of 3336 cylinders, as shown in Figure 7. Then you could format only the first volume by leaving out the -size and the -grow and let zFS dynamic secondary allocation allocate and format the additional volumes (up to 9 more) as needed. The IOEPRMxx aggrgrow configuration option must be on.

```
USERIDA JOB ,'Multi-Volume',
CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
DEFINE EXEC PGM=IDCAMS
SYSPRINT DD SYSOUT=H
SYSSDUMP DD SYSOUT=H
AMSDUMP DD SYSOUT=H
SYSSIN DD *
DEFINE CLUSTER (NAME(OMVS.VOL10.COMPAT.AGGR001) -
 VOLUMES(PRV000 PRV001 PRV002 PRV003 PRV004 -
 PRV005 PRV006 PRV007 PRV008 PRV009) -
 DATACLASS(EXTATTR) -
 LINEAR CYL(3336 3336) SHAREOPTIONS(3))
/*
//FORMAT EXEC PGM=IOEAGFMT,REGION=OM,
// PARM=('aggregate OMVS.VOL10.COMPAT.AGGR001 -compat')
//SYSPRINT DD SYSOUT=H
//STOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSSDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
/*@*
```

Figure 7. Example job to create a multi-volume compatibility mode aggregate, using a secondary allocation size
Adding a volume to a compatibility mode aggregate

To add a candidate volume to a zFS aggregate, use the IDCAMS utility ALTER command with the ADDVOLUMES parameter. Figure 8 shows an example job that adds two volumes to the (SMS-managed) OMVS.ZFS.AGGR1 zFS aggregate.

```
//SUIMGVMA JOB (ACCTNO), 'SYSPROG', CLASS=A,
// MSGCLASS=H, MSGLEVEL=(1,1), NOTIFY=&SYSUID
//STEP01 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=* 
//SYSIN DD *
//    DD * 
//    ALTER OMVS.ZFS.AGGR1.DATA -
//    ADDVOLUMES(* *)
/*
```

Figure 8. Example job to add volumes to a zFS aggregate

In this case, DFSMS is choosing the particular candidate volumes. If you want to specify the volumes, use their volume serials in place of the asterisks. See z/OS DFSMS Access Method Services Commands for more information about IDCAMS ALTER ADDVOLUMES. DFSMS states, if an ALTER ADDVOLUMES is done to a data set already opened and allocated, the data set must be closed, unallocated, reallocated, and reopened before VSAM can extend onto the newly added candidate volume.

For zFS, this means that if the zFS aggregate is already attached when the ALTER ADDVOLUMES is done, it must be detached and attached again before zFS can extend to the newly added candidate volume. Compatibility mode aggregates must be unmounted and mounted again (because that is when they are detached and attached). You can use the remount capability of z/OS UNIX. For details, see the topic on Remounting a mounted file system in z/OS UNIX System Services Planning.

Increasing the size of a compatibility mode aggregate

If your zFS file system runs out of space, you have several options to increase its size:

- You can grow the aggregate. For more information, see “Growing a compatibility mode aggregate” on page 27.
- If you cannot grow the aggregate (because, for example, there is no more room on the volume), you can add a volume to the aggregate. For more information, see “Adding a volume to a compatibility mode aggregate.”
- If you cannot grow the aggregate and you cannot add a volume (because, for example, you do not have any more volumes available), you can copy the aggregate into a larger VSAM linear data set. There are two ways to copy the data:
  - You can copy each file and directory of the zFS aggregate to a larger data set.
  - You can copy the physical blocks of the zFS aggregate to a larger data set.

Copying each file and directory of the aggregate to a larger data set

One method to increase the size of a zFS aggregate is to copy each file and directory of the aggregate to a larger data set. Figure 9 on page 32 shows an
This approach uses the `pax` command to copy the individual files and directories into an already formatted and empty zFS file system. Both file systems must be mounted. `pax` uses the z/OS UNIX file and directory APIs to read and write each individual file and directory of the hierarchy of the file system. (It does not copy lower mounted file systems because of the `-X` and `-M` options.) You can use the
ISHELL command or the automount command with the allocany or allocuser keyword to create the new larger aggregate to copy into with pax, because they format the aggregate.

If you are running this job on a system running z/OS V1R13 or later, and the file system was written to using a prior release of z/OS, zFS might use more DASD space for the same data than it did on the prior release. The increase in DASD space can occur for small files (1 KB in size or less) because beginning with z/OS VR13 zFS does not store data in 1-KB fragments; instead, it stores data in 8-KB blocks. For example, if the file system contained 1000 files that are 1 KB in size, on z/OS V1R13 zFS could use a maximum of 10 cylinders more than on previous releases. You can determine how many files are in the file system that are 1 KB or less using the following z/OS UNIX command:

```
find mountpoint -size -3 -type f -xdev | wc -l
```

After you successfully copy the data, when you are comfortable with the new, larger aggregate, you can delete the old aggregate.

**Copying the physical blocks of the aggregate to a larger data set**

Another method to increase the size of a zFS aggregate is to copy the physical blocks of the aggregate to a larger data set using the DFSMS REPRO command. This approach is normally faster than using the pax command. However, do not format the target zFS data set before using the REPRO command. Figure 10 on page 34 shows an example of this approach.
Figure 11 on page 35 shows a zFS file system (PLEX.OLD.AGGR002.LDS0002) that is full and a newly-defined zFS data set (PLEX.NEW.AGGR002.LDS0002 before the REPRO) that is larger. PLEX.NEW.AGGR002.LDS0002 has a larger HI-A-RBA than PLEX.OLD.AGGR002.LDS0002. When the blocks from PLEX.OLD.AGGR002.LDS0002 are copied into PLEX.NEW.AGGR002.LDS0002 using REPRO, the result is PLEX.NEW.AGGR002.LDS0002 after REPRO. There is now room to add data to PLEX.NEW.AGGR002.LDS0002.

Figure 10. Sample job to copy the physical blocks of an aggregate to a larger data set
With this approach, the new VSAM linear data set must not be formatted as an empty zFS file system before the REPRO command is used. (If the new data set was formatted, the REPRO would copy blocks to the end of the primary allocation, not the beginning. The data blocks being copied contain all the file system data and the file system information, so formatting is not necessary.) Neither file system needs to be mounted. REPRO uses native VSAM calls to read and write the blocks.

Follow these guidelines:

- When using the REPRO command, do not use the z/OS UNIX ishell command or the z/OS UNIX automount command with the allocany or allocuser keyword, because those commands will automatically format the aggregate.
- Do not use this approach to copy an HFS file system to a zFS file system because you will be copying the physical blocks of the file system (not the individual files) and the internal format of HFS file systems is different than the internal format of zFS file systems.

Notice that the ZFS attribute is not set in the LISTCAT output for the target data set (PLEX.NEW.AGGR002.LDS0002). It is set the first time the zFS file system is mounted read-write.

Now the new aggregate can grow into the available space in the allocated portion of the data set or even extend to additional extents if there is space on the volume.

After you successfully copy the data, when you are comfortable with the new, larger aggregate, you can delete the old aggregate.

---

**Figure 11. Copying blocks from a full zFS data set into a larger data set**

With this approach, the new VSAM linear data set must not be formatted as an empty zFS file system before the REPRO command is used. (If the new data set was formatted, the REPRO would copy blocks to the end of the primary allocation, not the beginning. The data blocks being copied contain all the file system data and the file system information, so formatting is not necessary.) Neither file system needs to be mounted. REPRO uses native VSAM calls to read and write the blocks.

Follow these guidelines:

- When using the REPRO command, do not use the z/OS UNIX ishell command or the z/OS UNIX automount command with the allocany or allocuser keyword, because those commands will automatically format the aggregate.
- Do not use this approach to copy an HFS file system to a zFS file system because you will be copying the physical blocks of the file system (not the individual files) and the internal format of HFS file systems is different than the internal format of zFS file systems.

Notice that the ZFS attribute is not set in the LISTCAT output for the target data set (PLEX.NEW.AGGR002.LDS0002). It is set the first time the zFS file system is mounted read-write.

Now the new aggregate can grow into the available space in the allocated portion of the data set or even extend to additional extents if there is space on the volume.

After you successfully copy the data, when you are comfortable with the new, larger aggregate, you can delete the old aggregate.
Decreasing the size of a compatibility mode aggregate

You can decrease the size of a zFS aggregate using the `pax` command. Figure 12 shows a sample job.

```plaintext
//SUIMGVMB JOB, 'SHRINK AGGR WITH PAX',
//   CLASS=A, MSGCLASS=X, MSGLEVEL=(1,1)
//* Make sure you have no line numbers in this JCL
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//SYSIN DD *
DEFINE CLUSTER (NAME(PLEX.NEW.AGGR002.LDS0002) - LINEAR CYL(25 5) SHAREOPTIONS(3) - VOLUMES(CFC000 CFC001))
/*
//FORMAT EXEC PGM=IOEAGFMT, REGION=0M,
//* On the next line, aggregate and compat must be lower case
//PARM=(''-aggregate PLEX.NEW.AGGR002.LDS0002 -compat')
//SYSPRINT DD SYSOUT=H
FLICT EXEC PGM=IKJEFT01, REGION=0M
SYSTSPRT DD SYSOUT=*
SYSEXEC DD DSN=SYS1.SBPXEXEC, DISP=SHR
SYSTSIN DD *
OSHELL /usr/sbin/mount -t ZFS -f PLEX.OLD.AGGR002.LDS0002 +
/service2 ; +
/service2 ; +
/pax -rwCMX -p eW . /service3 ; +
// The result of these next two steps should show that
// the new file system is smaller
//AGGRINF1 EXEC PGM=IOEZADM, REGION=0M,
// PARM=('aggrinfo PLEX.OLD.AGGR002.LDS0002 -long')
//SYSPRINT DD SYSOUT=*
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
/*
//AGGRINF2 EXEC PGM=IOEZADM, REGION=0M,
// PARM=('aggrinfo PLEX.NEW.AGGR002.LDS0002 -long')
//SYSPRINT DD SYSOUT=*
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
/*
```

Figure 12. Sample job to decrease the size of an aggregate

This approach uses the `pax` command to copy the individual files and directories into an already formatted and empty zFS file system. Both file systems must be mounted. `pax` uses the z/OS UNIX file and directory APIs to read and write each individual file and directory of the hierarchy of the file system. (It does not copy lower mounted file systems because of the -X and -M options.)
After you successfully copy the data, when you are comfortable with the new, smaller aggregate, you can delete the old aggregate.

**Renaming or deleting a compatibility mode aggregate**

To rename a compatibility mode aggregate, use the IDCAMS ALTER command with the NEWNAME parameter. You cannot rename an aggregate if it is mounted.

After the rename is done, the name of the file system stored in the zFS aggregate will not match the aggregate name. This is a requirement for compatibility mode zFS aggregates. To reconcile the file system and aggregate name, the zFS file system must be mounted initially as read/write after the IDCAMS RENAME is complete. This allows zFS to reconcile the file system name with the new aggregate name. After the name is reconciled, the aggregate can then be mounted read-only.

The example in Figure 13 assumes that:
- The data component name is the same as the cluster name with DATA appended
- You want to rename both the cluster name and the data component name.

```plaintext
//SUIMGVMS JOB (ACCTNO), 'SYSPROG', CLASS=A, // MSGCLASS=X, MSGLEVEL=(1,1), NOTIFY=&SYSUID //STEP01 EXEC PGM=IDCAMS //SYSPRINT DD SYSOUT=* //SYSIN DD * ALTER PLEX.JMS.AGGR006.LDS0006 - NEWNAME(PLEX.JMS.AGGR008.LDS0008) ALTER PLEX.JMS.AGGR006.LDS0006.* - NEWNAME(PLEX.JMS.AGGR008.LDS0008.*) /*
```

*Figure 13. Example job to reconcile the file system and aggregate name*

To delete a compatibility mode aggregate, use the IDCAMS utility DELETE command. You cannot delete an aggregate if it is mounted. Figure 14 shows a sample job that deletes both the cluster name and the data component.

```plaintext
//SUIMGVMD JOB (ACCTNO), 'SYSPROG', CLASS=A, // MSGCLASS=H, MSGLEVEL=(1,1), NOTIFY=&SYSUID //STEP01 EXEC PGM=IDCAMS //SYSPRINT DD SYSOUT=* //SYSIN DD * DELETE PLEX.JMS.AGGR006.LDS0006 /*
```

*Figure 14. Example job to delete a compatibility mode aggregate*

See [z/OS DFSMS Access Method Services Commands](https://www.ibm.com/support/knowledgecenter/POWER10/pf10_100000) for information and restrictions on IDCAMS ALTER NEWNAME and DELETE.
Changing zFS attributes on a mounted zFS compatibility mode file system

ZFS attributes are assigned to a zFS compatibility mode file system when it is mounted. The attributes can be set by specifying a zFS MOUNT PARM or they can be set from the zFS default values of the system where the primary mount occurs. These attributes, which are generally only meaningful for read/write mounted file systems, include the following:

- AGGRFULL
- AGGGRGROW
- CONVERTTOV5
- FSFULL

When one of these attributes is assigned to the file system when it is mounted, it typically remains with that file system until it is explicitly unmounted. However, if the file system's attributes were assigned from a zFS default set on the system, they may be changed in the following situations:

- The file system is NORWSHARE and z/OS UNIX ownership moves to another system with a different zFS default
- The file system is remounted samemode and the z/OS UNIX owning system has a different default
- The file system is remounted from read-only to read/write and the z/OS UNIX owning system has a different default
- The file system is NOAUTOMOVE and the system is coming up with a different default.

The RWSHARE and NORWSHARE attributes of a compatibility mode file system may also be changed if they were assigned from a zFS default of the system on which they were mounted.

For example, there are several cases when the RWSHARE attribute of a file system may be changed to NORWSHARE:

- The file system is remounted from read-only to read/write and the z/OS UNIX owning system has a NORWSHARE default
- The file system is NOAUTOMOVE and the system is coming up with a NORWSHARE default.

Similarly, if the NORWSHARE attribute was assigned from a zFS default, it may be changed to RWSHARE under the following situations:

- The file system has z/OS UNIX ownership moved to another system that has specified RWSHARE as the default
- The file system is remounted from read-only to read/write and the z/OS UNIX owning system has an RWSHARE default
- The file system is NOAUTOMOVE and the system is coming up with an RWSHARE default.

You can query the current default value of a zFS attribute by issuing the `zfsadm configquery` command. For example, to query the default value of the following attributes, you can issue the following commands:
You can change a zFS attribute on a mounted file system. To do so, take an appropriate action, as previously described for the attribute that you want to change. For example, to change the NORWSHARE attribute of a compatibility mode file system to RWSHARE, you can move the z/OS UNIX ownership of that file system to a different system that specifies RWSHARE as the zFS default.

Also, as the following examples show, you can change the zFS default values by issuing the `zfsadm config` command:

```bash
zfsadm config -aggrfull 95,5
zfsadm config -converttov5 on
zfsadm config -fsfull 90,10
zfsadm config -aggrgrow on
zfsadm config -sysplex_filesys_sharemode rshare
```

**Note:** Generally, to avoid getting unexpected attribute changes, it is best to have the zFS default values be the same on all members of the sysplex. However, if you want to change an attribute of a mounted file system, you can temporarily change a zFS default and then cause one of the situations previously described. For example, move the z/OS UNIX ownership of the file system to a different system where the zFS default has been temporarily changed, then change the default back to the original value. You can only change a zFS attribute of a mounted file system if you did not specify the attribute in a MOUNT PARM.

## Unmounting zFS file systems before copying or moving

When a user mounts (attaches) an aggregate to a particular system, zFS records the name of the system, the sysplex name (when it is a sysplex), and a time stamp in the zFS aggregate (in block zero of the aggregate). In addition, while the aggregate is mounted, zFS updates the time stamp every 30 seconds. If another system (that is not in the same sysplex) sharing the DASD attempts to mount the same aggregate, zFS on that system recognizes that the system name in the aggregate is not blank and does not match this system. In this case, zFS waits 65 seconds to see if the time stamp is updated (by the original system). If the time stamp is updated in that 65-second period, zFS refuses to mount the aggregate and returns ENXIO (X‘8A’) with reason code EF096058. As a result, zFS prevents a system from writing to a zFS aggregate that is mounted read/write on another system. If the time stamp is not updated, the mount succeeds after waiting for 65 seconds. A similar situation might occur when a copy was made of a zFS aggregate, or an entire DASD volume, while the zFS aggregates were mounted. In this case, when a mount is attempted of these ‘copies’, a 65-second block zero wait might be seen for each mount. This will be accompanied by an IOEZ00807I message that is issued by zFS.

When a zFS aggregate is unmounted (detached), the system name and the time stamp are cleared. In this case, the next mount does not wait because zFS knows that the aggregate is not currently mounted. If the aggregate is being mounted on a different member in the same sysplex after a failure, zFS does not wait because it recognizes that this is a different system that is in the same sysplex.
As a result, you can cause zFS to wait during mount unnecessarily and you can experience z/OS UNIX latch contention if you fail to unmount (detach) a zFS aggregate before copying it or moving it to another system.

### Understanding zFS disk space allocation

Unlike releases prior to z/OS V1R13, data is not stored in 1K fragments. Instead, the data is stored in 8K blocks. Releases z/OS V1R13 and later can read data stored in fragments; however, when the data is updated, it is moved into 8K blocks. Note that because previous releases of zFS can read an 8K block that is not full, no toleration support is required on those systems. Also, in previous releases, when zFS stored data in fragments, data from multiple files typically resided in separate 8K blocks.

However, there are certain cases when z/OS V1R13 and later will require more DASD space than zFS in previous releases. This can also occur if zFS running on V1R13 is in a mixed-release sysplex and becomes the owning system of a file system. For example, if every file in the file system is 1K or less, zFS z/OS V1R13 and later releases could require up to four times the DASD storage as previous releases. As a second example, because HFS uses 4K blocks to store data and zFS uses 8K blocks, if every file in the file system were 4K or less, zFS R13 could require up to twice as much DASD space to store these files. As another example, if the file system contained 1000 files that are 1K in size, zFS in z/OS V1R13 and later releases could take a maximum of 10 cylinders more than zFS in previous releases. Typically, however, any increase in the DASD storage used by zFS V1R13 and later releases will be negligible. For example, the R13 version root file system copied using zFS R13 takes approximately 2% more space than the same file system copied using zFS R11. Note that zFS releases z/OS V1R13 and later packs multiple ACLs and symbolic links into an 8K block which previous releases did not do.

To determine if an existing file system needs more DASD storage, you can use the “scan for small files” utility, **zfsspace**. For a mounted zFS file system, the utility shows the number of small files (1K or less), if a secondary allocation is specified, and if aggrgrow=on is specified. The **zfsspace** utility, along with other helpful tools, is available from the [z/OS UNIX System Services Tools and Toys web page](http://www.ibm.com/systems/z/os/zos/features/unix/bpxa1ty2.html). For more information about migration steps that might be needed, refer to [z/OS Migration](http://www.ibm.com/systems/z/os/zos/features/unix/bpxa1ty2.html).

Another result of moving fragments into 8-KB blocks is that the following situation can occur:

- A zFS file system is full, and
- It is zFS-owned on a V1R13 or later system, and
- It has no secondary allocation specified, or cannot extend because there is no space on the volume, and
- You try to remove some files in order to free up some space, but the remove fails due to return code ENOSPC (133)

This failure can occur because you are trying to remove an entry from a directory that was created before z/OS V1R13 and is smaller than 7 KB, so it is stored in fragments. But the file system is zFS-owned on a z/OS V1R13 or later system and needs a free 8-KB block to do the remove. To resolve this problem, you must explicitly grow the file system in order to make free 8-KB blocks available. You can do this even if the zFS file system data set does not have a secondary allocation size specified. Free space on the volume is required. For example:
If you need to add a volume, you can add one using the IDCAMS ALTER command with the ADDVOLUMES option. For more information, see “Adding a volume to a compatibility mode aggregate” on page 31.

A zFS aggregate is an array of 8-KB blocks. Three special objects are present in all zFS aggregates. These objects take up space in an aggregate, which means that space cannot be used for user files:

**Log file**
Records metadata changes. By default, its size is 1% of the disk size. However, it will never be smaller than 14 blocks and it will never be larger than 16,384 blocks (128 MB).

**Bitmap**
Lists the blocks that are free on disk. The file size is dependent on the size of the aggregate.

**Aggregate File System List**
Describes the file systems contained in the aggregate. For compatibility mode aggregates it is usually only one 8 KB block.

The `zfsadm aggrinfo` command shows aggregate disk space usage. This is based on the number of 8KB blocks. It subtracts the space reserved for the above three objects in its calculations (and tells you this in the output). The `zfsadm aggrinfo` command shows output in units of 1-KB blocks. If you use the `-long` option of the `zfsadm aggrinfo` command, it shows the number of free 8-K blocks, the number of free 1K fragments and the size (in K) taken up by the log file, the file system table and the bitmap.

The zFS threshold monitoring function `aggrfull` reports space usage based on total aggregate disk size. It incorporates the space for the above three special objects when showing total disk space and amount used on disk in its messages. The `aggrfull` message shows units in 8K blocks.

The `zfsadm aggrinfo` command shows the free space and the total aggregate size in 1-KB units.

The `df` command shows the file system free space, but because the `df` command shows things in 512-byte units, usually the `df` output for zFS is exactly twice the numbers shown for `zfsadm aggrinfo`.

zFS stores files on disk in one of three ways:

**Inline**
If the file is 52 bytes or less, it is stored in the same data structure on disk that holds the file status (such as owner, size, and permissions). A file 52 bytes or less takes no extra disk space.

**Fragmented**
On systems before z/OS V1R13, if the file is 7 KB or less and has never
been larger than 7 KB, zFS stores it in 1-KB fragments; as such, it is stored in part of an 8-KB block. Multiple small files can share the same 8 KB block on disk. On z/OS Releases z/OS V1R13 and later systems, zFS no longer stores files in 1-KB fragments.

**blocked**

On systems before z/OS V1R13, if the file is over 7 KB, it is stored in one or more 8-KB blocks. On Releases z/OS V1R13 and later systems, if a file is over 52 bytes, it is stored in one or more 8-KB blocks.

**How data is stored on systems before z/OS V1R13**

On systems before z/OS V1R13, zFS can store data in fragmented blocks to conserve disk space. On these systems, each small file does not need to use a full 8-KB block of disk space. However, as a result of this method of storing data, a problem can occur when data is stored using zFS. That is, the amount of free space that is displayed by the z/OS UNIX `df` command might not give the entire picture of free space. The `df -k` command displays free space in a file system in 1-KB units. In zFS, this space is a combination of full 8-KB blocks plus the free 1-KB fragments in fragmented blocks. For example, as Figure 15 shows, if there were two 8 KB blocks and twenty 1 KB blocks that are left, `df -k` reports 36 KB available.

![Disk space allocation example](image)

*Figure 15. Disk space allocation example 1*

Because this is a combination of 8-KB blocks and 1-KB blocks, it is possible that many 1-KB blocks are available but no 8-KB blocks remain. As shown in Figure 16 on page 43, for example, if there were 0 8-KB blocks left and 20 1-KB blocks available, `df -k` reports 20 KB available. If you try to create a 10 KB file, you might think that there is plenty of space. However, a 10-KB file is larger than 7 KB, and therefore uses full 8 KB blocks. Because there are no 8 KB blocks available, there is no room for a 10 KB file, even though there is 20-KB free space.
Other rules can further restrict how free space is used. A file that is 7 KB must be stored in 7 contiguous fragments. Therefore, even if there is 20 KB available in the file system, if there is no fragmented block with 7 contiguous 1-KB blocks available, the file system will report that there is no space for the file. Also, a file that is stored as fragments cannot share the same 8-KB block as a directory stored as fragments.

Fragments save disk space, but make space allocation more complicated. To provide the maximum options for space allocation, you need to have free 8-KB blocks. The `aggrfull` option of MOUNT and IOEFSPRM indicates the number of free 8 KB blocks. If you are out of 8-KB blocks, you will be limited in how much additional file space that can be allocated in the file system. You should grow the aggregate or allow it to be dynamically extended.

When a zFS compatibility mode aggregate becomes full, you can make more space available. This happens automatically if you have specified `aggrgrow` for the aggregate and you specified a secondary allocation size when you defined the aggregate (that is, the VSAM linear data set). You can increase the size of the aggregate with the `zfsadm grow` command. Of course, in each of these cases, you must have space available on the volume to extend into. Or, you might be able to erase some files from the file system to free up some space.

Note that because of the difference between how HFS and zFS manage disk space and block sizes, certain z/OS UNIX commands, such as `df` and `du` might display information differently.

---

**Sharing zFS data in a non-shared file system sysplex**

For information about sharing zFS data in a shared file system in a multisystem sysplex environment, see Chapter 5, “Using zFS in a shared file system environment,” on page 51 and review “Unmounting zFS file systems before copying or moving” on page 39.

The only fully supported way to share zFS data between systems in a non-shared file system sysplex environment is read-only sharing, where a zFS file system is mounted read-only to each system. Results are undefined when a zFS file system is mounted read/write to one system and mounted read-only on another.
Minimum and maximum file system sizes

The minimum zFS compatibility mode aggregate size is six 3390 tracks, which hold thirty-six 8 KB blocks (six 8 KB blocks per track x 6 tracks). In the example in Figure 17 DFSMS allocates 7 tracks. Six 8-KB blocks per track x 7 tracks is 42 8-KB blocks or 336 KB. This only leaves 184 KB of free space available for files and directories. Small file systems tend to fill up quickly because of block and fragment allocation and can appear to have free space when they really do not. (For more information, see “Understanding zFS disk space allocation” on page 40). Using such small file systems is not a good idea. You can permit the file system to grow automatically (you must have aggregrow=on in the IOEFSPRM file, which is the default, or in the MOUNT PARM. You must also have a secondary allocation specified on the zfsadm define command, which is specified as 5 in Figure 17). However, your log file size is very small and might cause contention. The log file size cannot be increased after the aggregate is formatted.

Version 1.5 aggregates

For a version 1.5 aggregate, the architected maximum size for compatibility mode aggregates is approximately 16 TB (4 KB x 4 GB). If you use 3390 DASD that has 262,668 cylinders per volume, you can create a compatibility mode aggregate of about 11,425,931,919,360 bytes.

Version 1.4 aggregates

For a version 1.4 aggregate, the architected maximum size for compatibility mode aggregates is approximately 4 TB (1 KB x 4 GB). If you use 3390 DASD that has 65,520 cylinders per volume, you can create a compatibility mode aggregate of about 2,850,088,550,400 bytes.
Restriction: A zFS version 1.4 compatibility mode aggregate is limited to 4 TB even on extended address volume (EAV) devices. A zFS version 1.5 compatibility mode aggregate is limited to 16 TB even on extended address volume (EAV) devices.

The maximum number of objects (files, directories, and ACLs) in a zFS file system is 4 G. The maximum size of a file is approximately 4 TB. The maximum size of a directory is 4 GB. There is a limit of 65,533 (64K -1) subdirectories in a directory for a v4 directory. There is a limit of 4,294,967,293 (4G-1) subdirectories in a directory for an extended (v5) directory. The maximum number of names in a directory is dependent on the length of the names. However, there is a known performance problem when you have a large number of names (hundreds of thousands or millions) in a single zFS v4 directory. For best performance, use an extended (v5) directory in a version 1.5 aggregate. See “Using version 1.5 aggregates and extended (v5) directories” on page 24 for information about extended (v5) directories. If you must use a version 1.4 aggregate because you are still running releases prior to z/OS V2R1, try to spread names among many directories.

Caution: Do not use version 1.5 aggregates until you are sure you will not run any releases before z/OS V2R1.

v4 directory considerations

This section applies only to a v4 directory.

If you have long response times, you can get a first indication whether you might have a directory size problem by examining the output of the MODIFY ZFS,QUERY,KN operator command or the z/OS UNIX zfsadm query -kpfs command. Look at the Avg Time field on the lines for operations that require zFS to search through names of a directory (for example, zfs_lookup, zfs_create, or zfs_remove). Typically, the average times should be on the order of a few milliseconds. If they are relatively large (perhaps ten to a hundred times larger than that), it is possible that you have a directory that is too large and is causing performance problems. You can use the largedir.pl command to help determine which directories are large. It reports any zFS directories that have a large size. The largedir.pl command is available on the z/OS UNIX System Services Tools and [tools web page](http://www.ibm.com/systems/z/os/zos/features/unix/bpxa1ty2.html).

To determine how large a particular directory is (how many bytes the directory contains), use the ls -ld command against the directory to display its size in bytes. For example, if you suspect /zfsmnt5/testdir is too large, issue a command similar to the following one:

```
# ls -ld /zfsmnt5/testdir
drwxr-xr-x  2 G0DOUG AUDIT 1638400 Jan 18 2007 /zfsmnt5/testdir
```

The output shows /zfsmnt5/testdir is over 1 MB in size and contains many names (or at one time contained many names).
Space is not reclaimed when names are removed from a v4 directory. Therefore, you must look at the size of the directory rather than the number of names it currently contains. To reclaim the space, you can remove the directory rather than erasing names within it, or you can convert it to an extended (v5) directory. So if the directory currently has few names, but is large, try using either one of the following sets of commands to make a new directory:

```
mkdir /zfsmnt5/testdir2
cp /zfsmnt5/testdir/* /zfsmnt5/testdir2
rm -r /zfsmnt5/testdir
mv /zfsmnt5/testdir2 /zfsmnt5/testdir
```

Or:

```
mkdir /zfsmnt5/testdir2
/samples/copytree /zfsmnt5/testdir /zfsmnt5/testdir2 (if testdir has subdirectories)
rm -r /zfsmnt5/testdir
mv /zfsmnt5/testdir2 /zfsmnt5/testdir
```

Or:

```
zfsadm convert -path /zfsmnt5/testdir
```

If the large directory had mount points contained in it, you must unmount those file systems and mount them onto the mount points in the new directory before removing the large directory.

If the large directory is the root directory of a file system, you cannot remove it. You have two options:

- Copy the file system to another (new) file system and delete the original file system, or
- Convert the file system to a version 1.5 file system

See Chapter 7, “Migrating data from HFS to zFS,” on page 67 for information about copying one file system to another. For information about converting an existing file system to version 1.5, see “Using version 1.5 aggregates and extended (v5) directories” on page 24.

When you must have many file names in a single directory, it is best to use a version 1.5 directory for that application.

To help alleviate this large directory performance problem for a limited amount of time (z/OS V1R13 and z/OS V2R1 only) for some larger directories before migrating to version 1.5 aggregates, zFS will allow the creation of new Large Fast Lookup Cache buffers above the bar (64-bit storage) that will be used to fully cache large directories. Directories that are eligible to use this new cache must be small enough to fit in the cache and yet large enough to meet minimum size requirements.

This partial solution might help in certain cases where the size of a frequently accessed directory in a zFS aggregate has become too large and its performance is degraded. This is a global setting and is not specific to any one zFS aggregate or directory. All zFS directories that meets the criteria defined by the setting of the new IOEPRMxx configuration option flc is eligible to use this new cache. The user
must be aware of their file system environment and directory sizes in order to
choose the proper values for the flc parameters or they will not realize the
maximum benefit of this new function.

These buffers include a hash table for faster directory searching. There is also
storage allocated below the bar to map the free space in the directories cached in
these large buffers and also to cache the first level indirect block. This is additional
storage that zFS will need to use, so administrators will have to determine if there
is enough real storage and paging space to accommodate the needed buffers above
the bar, and also that the space exists in the ZFS address space to accommodate
these large buffers (approximately 10K for each large FLC buffer). You can see how
much storage the ZFS address space is currently using by issuing the modify
zfs,query,storage command. Compare the third line of data (USS/External
Storage Access Limit) and fourth line of data (Total Bytes Allocated
(Stack+Heap+OS)) to see approximately how much storage you have available for
use.

This solution might not be acceptable for all installations. For example, if there are
hundreds of very large directories, then there may not be enough storage available
in the ZFS address space or in paging data sets or auxiliary storage to make this a
viable option. If the large directories are not frequently accessed, the benefits will
be minimal. The administrator should take all of these factors into account before
electing to use Large FLC buffers. If you do elect to use the Large FLC buffers,
then you should periodically monitor your performance and your directory sizes to
ensure the Large FLC buffer definition is properly configured for your
environment.

Large FLC buffers are not created by default. To get them, the administrator must
specify the new IOEPRMxx configuration option flc. For more information about
the option, see “IOEFSRPM” on page 193. Following is the general format of the
flc option.

flc=mindirsize,bufsize,bufnum,inactive

For example:

flc=256K,8M,30,45

The mindirsize parameter

The mindirsize parameter is the minimum directory size that will be processed
using the large FLC buffers. The Large FLC buffer is used for the first directories
accessed that match or exceed the mindirsize value specified (if the large buffer is
large enough to contain the entire directory). And, it will remain associated with
this directory until either the vnode is recycled or the inactive time period is
observed for directory access. You can determine the size of a directory by using
the ls -l command. See information in the beginning of this section for further
assistance in determining directory sizes. An example of output from the ls -l
command is:

drwxr-xr-x+ 2 STC1 AUDIT 303104 May 8 00:43 ftestDS

drwxr-xr-x+ 2 STC1 AUDIT 303104 May 8 00:51 ftestDS2

drwxr-xr-x+ 2 STC1 AUDIT 1302528 May 8 01:32 ftestMS

If you wanted to consider all three directories large, then you would set mindirsize
as the smallest of the three (or less), 303104 or 296 K. If you want to consider
ftestMS as large but not the other two, then you would select a mindirsize between
303105 and 1302528, such as 1 M.
The bufsize parameter

The bufsize parameter is the amount of storage above the bar that will be allocated for each Large FLC buffer. The size needed is based on your largest directory. Take into consideration future growth when determining how many directory entries there will be. Directories that are too large for the large FLC buffer will not be eligible to use them.

The bufsize value is calculated how many directory entries your largest directory will hold, and how many characters are in the entry names. You can determine the maximum number of entries a directory can hold by using the size of the directory returned from the ls -l command. Each directory block is 8K and can hold at most 250 entries. Future growth of the directory should also be considered to determine how many directory entries there will be. The Large FLC buffer will be broken up into pieces that will hold the directory entries. Each piece is 64 bytes in length. If the entry name will not fit in one piece, then secondary pieces will be used. The first piece has space for 40 characters and secondary pieces will allow 52 more characters each. The bufsize specification should then be 64 times the number of needed pieces. In the example above, the maximum number of entries the directory ftestMS can have is \((1302528/8192) \times 250 = 39750\). So, a minimum bufsize value would be 39750 * 64 = 2484K. If you expect the directory to double in size in the future, then you should set bufsize to 39750 * 2 * 64 = 4968K.

The bufname parameter

The bufnum parameter should be the total number of large directories needed, that is, those with sizes greater than or equal to mindirsize. It is the number of Large FLC buffers that zFS is to allocate and use. These buffers are obtained during zFS initialization and queued to a free list until needed. When a large directory is accessed, one will be obtained from this free list and assigned to that directory (until the directory is deleted or inactive for inactive seconds or recycled for use by another directory).

The inactive parameter

The inactive parameter defines the interval of inactivity before a Large FLC buffer can be stolen from one large directory and made available for another. The value is specified as a number of seconds. The default is 30.

Other IOEFSPRM changes to consider when using large FLC buffers

If the Large FLC buffers are to be used, then it is recommended that, if possible, for maximum performance gain, that the IOEPRMxx configuration option meta_cache_size (or metaback_cache_size) be increased by the sum of the sizes of all of the large directories being cached in the Large FLC buffers. And, the IOEPRMxx configuration option vnode_cache_size should be increased by the sum of the number of entries in the large directories being cached in the Large FLC buffers. You should verify with the modify zfs,query,storage report that the zFS address space has enough room to increase these caches.

Normally, it is beneficial for applications that constantly create and destroy large version 4 directories to also remove the directory. Removing the direction is helpful because the space on disk that the directory blocks use is not reclaimed until the directory itself is removed. If the directory blocks were not removed, then creating new entries in the directory would require reading each empty directory block to
verify the new entry does not already exist. If you elect to use the Large FLC buffers, it might be better to not remove an empty large directory from disk. This is because zFS will immediately recognize that the size of the directory is above the IOEPRMxx variable \textit{fcl mindirsize} setting, and assign a Large FLC Buffer to it upon first access. This makes future creates much faster because there will be no need to repeatedly read in the empty directory blocks to verify that the new entry does not exist.
Chapter 5. Using zFS in a shared file system environment

zFS supports a shared file system capability in a multisystem sysplex environment. The term *shared file system environment* refers to a sysplex that has a BPXPRMxx specification of SYSPLEX(YES). That is, users in a sysplex can access zFS data that is owned by another system in the sysplex. For full sysplex support, zFS must be running on all systems in the sysplex in a shared file system environment.

**Tip:** To better understand the terminology and concepts in this section, review “Terminology and concepts” on page 4.

Overview of the shared file system environment

You can enable zFS to run sysplex-aware for read/write mounted file systems as well as for read-only mounted file systems. For details, see “Applying required APARs for z/OS V2R1” on page 14.

Beginning with z/OS V1R13, zFS runs sysplex-aware on a file system basis (sysplex=filesys). That is, a system running zFS R13 or later in a shared file system environment is always capable of mounting zFS read/write file systems as sysplex-aware. The default is to mount all zFS read/write file systems as non-sysplex aware. However, you can specify that you want any individual zFS read/write file system to be sysplex-aware in one of two ways:

- You can specify the RWSHARE MOUNT PARM.
- You can specify the sysplex_filesys_sharemode=rwshare zFS configuration option in your IOEFSPRM file. This option sets the default to be that all zFS read/write file systems are sysplex-aware, unless you specify a MOUNT PARM of NORWSHARE to make a specific file system non-sysplex aware.

Beginning with z/OS V1R13, if you specify sysplex=on in your IOEFSPRM file, zFS runs with sysplex=filesys; however, it internally sets the sysplex_filesys_sharemode value to rwshare (if you did not explicitly specify a different sysplex_filesys_sharemode value in your IOEFSPRM file). This behavior makes zFS read/write mounted file systems sysplex-aware by default. You should change your sysplex specification to sysplex=filesys, if you need to bring a V1R12 system back into the shared file system environment. You should also specify sysplex_filesys_sharemode=rwshare, if you want zFS read/write file systems to be sysplex-aware by default.

The following sections describe how the shared file system environment works using various configurations and the commands for determining the file system owner.

Read-only mounted file systems

When a file system is mounted read-only (such as on SY2), the mount request is sent to the local physical file system (in this case, zFS) and zFS opens the file system data set (for read). If the mount is successful on that system, z/OS UNIX records the mount and sends a signal to the other sysplex member systems to issue a “catch-up” mount on each system. Each z/OS UNIX on each other system then reads the couple data set (CDS) and determines that it needs to send a mount request to the local zFS for that file system. Each “local mount” causes zFS to open
the data set (for read). In this way, the mount on SY2 causes the file system to be mounted on every member of the sysplex.

For read-only mounted file systems, file requests are sent directly to the local physical file system, which directly reads the file system data on DASD (see Figure 18). That means each zFS on each system has the zFS file system opened (for read) and directly accesses the data. Read-only mounted file systems are referred to as being *sysplex-aware*.

### zFS support for read/write non-sysplex aware mounted file systems

This section applies to file systems that are non-sysplex aware (see Figure 19).

---

**Figure 18. Sysplex-aware file system (read-only)**

For read-only mounted file systems, file requests are sent directly to the local physical file system, which directly reads the file system data on DASD (see Figure 18). That means each zFS on each system has the zFS file system opened (for read) and directly accesses the data. Read-only mounted file systems are referred to as being *sysplex-aware*.

**Figure 19. Non-sysplex aware file system (read/write)**

When a file system is mounted read/write (for example, on SY2), the mount request is sent to the local physical file system (in this case, zFS) and zFS opens the file system data set (for read/write). If the mount is successful on that system, z/OS UNIX sends a signal to the other sysplex member systems to issue a catch-up mount on each system. Each z/OS UNIX on each other system then reads the CDS and determines that it needs to record that the file system is mounted.
read/write and that SY2 is the owning system. The local zFS on the other systems does not receive a local mount request. The mount on SY2 is “logically” mounted on each of the other systems.

For read/write mounted file systems, file requests are function shipped to the owning system. The owning system is the only system where the file system is locally mounted and the only system that does I/O to the file system. File requests from systems other than the owning system have a longer pathlength through XCF communications. Read/write mounted file systems are referred to as being non-sysplex aware.

zFS support for read/write sysplex-aware mounted file systems

You can specify that some (or all) of your zFS read/write file systems should be sysplex-aware. For details about when this is appropriate, see “Specifying zFS file systems as sysplex-aware” on page 15. When a file system is mounted read/write sysplex-aware (for example, on SY2), the mount request is sent to the local zFS and zFS opens the file system data set (for read/write). If the mount is successful on that system, z/OS UNIX sends a signal to the other sysplex member systems telling them to issue a catch-up mount on each system. Each z/OS UNIX on each other system then reads the CDS and determines that it needs to send a mount request to the local zFS for that file system. Each local mount causes zFS to open the data set (for read/write). In this way, the mount on SY2 causes the file system to be mounted on every member of the sysplex.

For read/write mounted file systems that are sysplex-aware, file requests are sent directly to zFS, which then function ships the request to the zFS owning system. In many cases, zFS does not need to function ship the request to the owning zFS system because the data is in the local cache. zFS caches data to avoid frequent forwarding to an owner; typically, this improves performance for most workloads.

zFS support for read/write file systems with different levels of sysplex-awareness

zFS allows individual zFS read/write file systems to be mounted sysplex-aware or non-sysplex aware. During mount processing, the sysplex-awareness of an individual zFS read/write file system can be controlled by the value that is specified on the mount PARM for that file system or by the
The *sysplex_filesys_sharemode* option that is specified in IOEFSPRM summarizes how the sysplex awareness is determined.

<table>
<thead>
<tr>
<th>MOUNT PARM</th>
<th>Resulting awareness of the zFS read/write file system</th>
</tr>
</thead>
<tbody>
<tr>
<td>RWSHARE</td>
<td>Sysplex-aware</td>
</tr>
<tr>
<td>NORSHARE</td>
<td>Non-sysplex aware</td>
</tr>
<tr>
<td>None specified</td>
<td>Determined by the value, if any, specified on the <em>sysplex_filesys_sharemode</em> option.</td>
</tr>
</tbody>
</table>

- **rwshare**: File system is sysplex-aware
- **norwshare**: File system is non-sysplex aware
- **not specified**: File system defaults to be non-sysplex aware

Figure 21 shows one file system that is mounted NORSHARE and the other mounted RWSHARE. They are both z/OS UNIX owned on SY2. The NORSHARE file system is a non-sysplex aware file system; it is only locally mounted on the z/OS UNIX owner and requests from z/OS UNIX clients are function shipped to the z/OS UNIX owner by z/OS UNIX. A `df -v` command for the NORSHARE file system (FS1) from SY1 would display `Client=Y`. The other file system is mounted RWSHARE. It is a sysplex-aware file system; it is locally mounted on all systems and z/OS UNIX does not normally function ship requests to the z/OS UNIX owner. A `df -v` command for the RWSHARE file system (FS2) from SY1 would display `Client=N`.

```
MOUNT FILESYSTEM('OMVS.PRIV.COMPAT.AGGR001') TYPE(ZFS) MODE(RDWR)
MOUNTPOINT('/usr/mountpt1') PARM('RWSHARE')
```
**zFS enhanced sysplex-aware support**

Beginning in z/OS V1R13, zFS provides enhanced sysplex-aware support. When a zFS read/write file system is mounted sysplex-aware in a shared file system environment where all systems are running z/OS V1R13 or later, zFS can directly read and write zFS data from all of the V1R13 or later systems. If both the owning system and the requesting system are running z/OS V1R13 or later (and the file system is sysplex-aware), zFS directly accesses the file system. While zFS data is directly read and written, zFS metadata is normally read and written through the zFS owning system (SY2 in Figure 22). In some cases, zFS metadata can be directly read. If the zFS owning system or the requesting system are running a release of z/OS earlier than z/OS V1R13, zFS function ships requests to the zFS owning system.

![Figure 22. zFS sysplex-aware file system being directly accessed from all z/OS V1R13 systems](image)

**Figure 22** is similar to **Figure 21** on page 54; however, all systems (SY1, SY2 and SY3) are running z/OS V1R13 or later. In this case, the sysplex-aware file system (FS2) is zFS owned by SY2 and user data is being directly accessed from all three systems.

There are some cases when an application running on a system (SY1) that is doing direct I/O can be affected by problems on the zFS owning system (SY2) such as a failing system or having I/O failures on the owning system during metadata updates. The application can also be affected if it needs to traverse a higher level directory contained in a file system that is owned by the failing system.

**zFS ownership versus z/OS UNIX ownership of file systems**

For zFS read/write sysplex-aware file systems, zFS takes responsibility for determining how to access the data. This means that zFS must have the concept of a file system owner to coordinate file requests. That system is the **zFS owner**. z/OS UNIX has its indication of owner, which is called the **z/OS UNIX owner**. The zFS owner is independent of the z/OS UNIX owner. The zFS owner is the system that coordinates file access. The z/OS UNIX owner generally does not have any performance implications when zFS runs sysplex-aware because file requests are sent to the local zFS rather than being function shipped to the z/OS UNIX owner. There are some cases when the z/OS UNIX owner is relevant (see "When is the z/OS UNIX owner important?" on page 58).
In Figure 20 on page 53, SY2 is the z/OS UNIX owner and the zFS owner. This is typically the case for the system where the mount was issued. If SY2 goes down, a new zFS owner is chosen randomly (such as SY3) and a new z/OS UNIX owner is chosen randomly (such as SY1) assuming it was mounted with AUTOMOVE. Figure 23 shows the situation after SY2 has come back up. (zFS on SY1 communicates directly with zFS on SY3.) The fact that SY1 is the z/OS UNIX owner is not important for performance in this case.

Note: For zFS non-sysplex aware file systems, the z/OS UNIX owner and the zFS owner are always the same system.

**Determining the file system owner**

To determine the zFS owner of a zFS file system, use the `zfsadm lsaggr` command. To determine the z/OS UNIX owner, use the following commands:

- `df -v` shell command
- `d omvs,f` operator command
- `f bpxoinit,filesys=d,all` operator command

Figure 24 shows the output of the `zfsadm lsaggr` command and the `df -v` command after the file system was mounted (as shown in Figure 20 on page 53) issued from SY2:

```
# zfsadm lsaggr
IOEZ00106I A total of 1 aggregates are attached
PLEX.JMS.AGGR008.LARGE08 SY2 R/W

# df -v
Mounted on Filesystem Avail/Total Files Status
/zfsmnt5 (PLEX.JMS.AGGR008.LARGE08) 2853944/3745440 Available
ZFS, Read/Write, Device:26, ACLS=Y
File System Owner : SY2 Automove=Y Client=N
Filetag : T=off codeset=0
Aggregate Name : PLEX.JMS.AGGR008.LARGE08
```

**Figure 24. zfsadm lsaggr and df -v output after mount**

Figure 25 on page 57 shows the output of the `d omvs,f` command after the file system was mounted.
Figure 26 shows the output of the `zfsadm lsaggr` command and the `df -v` command after the file system was moved (as shown in Figure 23 on page 56) by both z/OS UNIX and zFS and SY2 has come back up. The `zfsadm lsaggr` and `df -v` commands are issued from SY2:

```sh
# zfsadm lsaggr
IOEZ00106I A total of 1 aggregates are attached
PLEX.JMS.AGGR008.LARGE08 SY3 R/W

# df -v
Mounted on Filesystem Avail/Total Files Status
/zfsmnt5 (PLEX.JMS.AGGR008.LARGE08) 2853944/3745440 4294917290 Available
ZFS, Read/Write, Device:26, ACLS=Y
File System Owner : SY1 Automove=Y Client=N
Filetag : T=off codeset=0
Aggregate Name : PLEX.JMS.AGGR008.LARGE08
```

Figure 27 shows the output of the `d omvs,f` operator command after the file system was moved. Notice two important points:

- The zFS owner (SY3) and the z/OS UNIX owner (SY1) are different.
- The last `df -v` command reports that SY2 is not a client, even though SY2 is not the z/OS UNIX owner.

This situation occurs because the zFS file system is sysplex-aware and file requests are not function shipped by z/OS UNIX. Rather, the file requests are handled by zFS and metadata updates are sent to the zFS owner. Each local catch-up mount causes zFS to open the file system data set for read/write, and each system is prepared to read and write the file system. Because the file system is opened on each system, each system prepares to take ownership of the file system if that becomes necessary.

**Tip:** You can use the DISPLAY GRS system command to determine the zFS owner of a zFS file system. Use the RNAME for either the read-only or read/write file
system. For example, issue the following command to display the system name of the zFS owner as the exclusive owner of the resource name.

D GRS,RES=(SYSZIOEZ,IOEZLT.file_system_name)

For more information, see the serialization summary and list of ENQs in z/OS MVS Diagnosis: Reference.

When is the z/OS UNIX owner important?

The z/OS UNIX owner is important when a zFS read/write file system is non-sysplex aware. In this case, all file requests are handled through z/OS UNIX function shipping to the z/OS UNIX owning system. The z/OS UNIX owner and the zFS owner are always the same system.

When a zFS sysplex-aware file system is mounted, z/OS UNIX causes the file system to be locally mounted on each system (where zFS is running sysplex-aware). These are called catch-up mounts. If a local catch-up mount fails (for example, because the DASD is not accessible from that system), then z/OS UNIX treats that system (such as SY1) as a client and function ships requests to the z/OS UNIX owner (SY2). The system (SY1) might issue message BPXF221I. In this case, a df -v command issued from SY1 indicates Client=Y for that file system. In turn, zFS directly accesses the file system and function ships metadata updates to the zFS owner, if the zFS owner is a different system than the z/OS UNIX owner—in this case, it is not different (for example, see Figure 28).

The zFS owner can be different than the z/OS UNIX owner. In this case, the request is function shipped by z/OS UNIX (from SY1) to the z/OS UNIX owner (SY2) and then is handled by direct access to the file system. Metadata updates will be function shipped by zFS to the zFS owner.

Similarly, if a local mount fails in the read-only mount case, z/OS UNIX treats that system as a client and function ships (the read) requests to the z/OS UNIX owning system. zFS does not typically function ship in the read-only case regardless of which system is the zFS owner.

Figure 28. File system ownership when mount fails
Dynamic movement of the zFS owner

For zFS read/write sysplex-aware file systems, an important aspect of performance is which system is the zFS owner. The zFS owner is the system that does metadata updates to the file system. zFS automatically moves the zFS owner among zFS systems, based on the amount of activity at the zFS owner from each system. The frequency of the dynamic ownership movement can vary, depending on the level of zFS. On z/OS V1R13 and later systems, ownership moves less often than on systems running previous levels of z/OS. In addition, if an aggregate is read/write sysplex-aware and zFS-owned on a z/OS V1R13 or later system, zFS does not dynamically move ownership to a V1R12 system, even if it is running sysplex=filesys or sysplex=on.

File requests do not fail as a result of dynamic aggregate movement. New requests are suspended until the aggregate is moved and then requests are allowed to complete. The system produces the following messages (for example):

Source system
22.19.12 DCEIMGVM IOEZ00548I Requesting that DCEIMGVM takeover aggregate PLEX.JMS.AGGR006.LDS0006 LDS0006 (requests: local 2, new owner 1202 total 1204

Target system
22.19.12 DCEIMGVM IOEZ00388I Aggregate takeover being attempted for aggregate PLEX.JMS.AGGR006.LDS0006
22.19.12 DCEIMGVM IOEZ00044I Aggregate PLEX.JMS.AGGR006.LDS0006 attached successfully.

In message IOEZ00548I, local requests is the number of requests on the source system during the measurement period. New owner requests is the number of requests from the target system during the measurement period. Total requests is the total number of requests from all systems during the measurement period. (Total requests can be greater than the sum of the local requests and the new owner requests). This information is provided to aid in problem determination.

For zFS sysplex-aware file systems, zFS aggregate movement is essentially independent of z/OS UNIX ownership movement (except for the cases discussed later in this section). When z/OS UNIX ownership movement occurs because of the MOUNT AUTOMOVE specification (for example, AUTOMOVE or AUTOMOVE(INCLUDE,SY1,SY2) or AUTOMOVE(EXCLUDE,SY1,SY2)), the z/OS UNIX ownership movement is as expected. Because z/OS UNIX sends requests directly to the local zFS, the z/OS UNIX ownership movement does not change the way that the zFS aggregate is accessed. z/OS UNIX ownership movement between zFS sysplex-aware file systems that have local mounts does not change how the file system is accessed.

There are several cases where the AUTOMOVE option of z/OS UNIX does change file system access:

NOAUTOMOVE
When this option is used, z/OS UNIX makes the file system unavailable (unowned). This causes any file access to be denied by z/OS UNIX.

UNMOUNT
When this option is used, z/OS UNIX unmounts the file system (across the sysplex). This causes the file system to be unmounted and any access occurs on the underlying file system.

Guideline: Mount system-specific zFS file systems with UNMOUNT instead of NOAUTOMOVE.
One way to think of the relationship between z/OS UNIX ownership movement and zFS aggregate ownership movement is:

- z/OS UNIX controls whether there is any access at all
- zFS ownership controls which system updates the metadata.

If a zFS read/write file system is non-sysplex aware, then z/OS UNIX controls movement of zFS read/write mounted file systems as in prior releases for a shared file system environment and the z/OS UNIX owner and the zFS owner are always the same.

## Using zFS in a shared file system environment

The following considerations apply when using zFS in a sysplex in shared file system mode:

- The file system hierarchy appears different when viewed from systems with zFS mounted file systems than it does from those systems not running zFS. The path name traversal through zFS mount points have different results in such cases because the zFS file system is not mounted on those systems not running zFS.
- zFS file systems that are owned by another system are accessible from a member of the sysplex that is running zFS.
- zFS compatibility mode file systems can be automoved and automounted. A zFS compatibility mode file system can only be automoved to a system where zFS is running.
- To share IOEFSPRM across a sysplex, the following specifications must use system symbols to differentiate the data set names:
  
  - `trace_dsn`
  - `msg_output_dsn`

In this case, you should use the `&SYSNAME` system variable in the IOEZPRM DD of the ZFS PROC to specify a different IOEFSPRM for different systems.

If you are not specifying a `msg_output_dsn` or a `trace_dsn` (or you can use system symbols), and you use the same options for all ZFS PFSs on all systems, you can share the same IOEFSPRM across systems.

If you want to share IOEFSPRM and you want to specify data set names in IOEFSPRM, you might be able to use system symbols. For example, if you have sysplex member systems SY1 and SY2, and you have allocated trace data sets named USERA.SY1.ZFS.TRACE and USERA.SY2.ZFS.TRACE, you can specify `trace_dsn=USERA.&SYSNAME..ZFS.TRACE` in your shared IOEFSPRM.

As a preferred alternative to the IOEZPRM DDNAME specification, the IOEFSPRM member can be specified as a true PARMLIB member. In this case, the member has the name IOEPRMxx, where xx is specified in the parmlib member list. It is possible to have multiple IOEPRMxx members and it is also possible to have an IOEPRMxx member that are shared among all members of the sysplex and another IOEPRMxx member that contains options that are specific to a particular sysplex member. See "IOEFSPRM" on page 193 for more information about IOEPRMxx.

The following information describes z/OS UNIX considerations when some or all systems are running zFS:

- All systems running zFS see zFS compatibility mode file systems. The file system hierarchy appears differently when viewed from systems with zFS mounted compatibility mode file systems than it does from those systems that are not running zFS. The path name traversal through zFS mount points have
different results in such cases because the zFS compatibility mode file system is not mounted on those systems that are not running zFS.

- If a system running zFS is brought down:
  - zFS compatibility mode file systems owned by the system that can be automoved are automoved to another system running zFS. If this function fails to find another owner, the file system becomes unowned. IBM recommends mounting zFS file systems with UNMOUNT instead of NOAUTOMOVE.
  - zFS compatibility mode file systems that are NOAUTOMOVE, become unowned.
  - zFS compatibility mode file systems that are unowned are not visible in the file system hierarchy, but can be seen from a D OMVS,F command. To recover a zFS compatibility mode file system that is mounted and unowned, the zFS compatibility mode file system must be unmounted.
  - The unowned zFS compatibility mode file systems can be recovered if the original owning system is brought back into the sysplex.

- If zFS is brought down on one system in the sysplex:
  - zFS compatibility mode file systems owned by the system that can be automoved are automoved to another system running zFS. If this function does not find another z/OS UNIX owner, the zFS compatibility mode file system, and all file systems mounted under it, are unmounted in the sysplex.
  - zFS compatibility mode file systems that are NOAUTOMOVE and, all file systems mounted under them, are unmounted in the sysplex.
  - When zFS is down on one system (SY1) in the sysplex, z/OS UNIX does not function ship any zFS compatibility mode file system that is subsequently mounted on another system. That file system is not visible from SY1. zFS can be brought up again on that system by responding R to the BPXF032D prompt. When this occurs, mounted file system visibility is established by one of the following methods:
    - If the zFS file system is non-sysplex aware, z/OS UNIX function shipping is established
    - If zFS file system is sysplex-aware, the zFS file system is locally mounted
  - When a zFS is brought down after a compatibility mode file system is mounted, the file system either continues to be function shipped or becomes function shipped. When zFS is brought back up on that system, the file system either:
    - Continues to be function shipped, when the zFS file system is non-sysplex aware
    - Is locally mounted, when the zFS file system is sysplex-aware

*zfsadm* commands work across the shared file system environment. You can display and modify zFS compatibility mode aggregates and file systems using *zfsadm* from any member of the sysplex, regardless of which member owns the aggregate.
Chapter 6. Performing a backup of zFS

This section describes how to back up a zFS aggregate using a DFSMSdss logical dump. DFSMSdss automatically performs a quiesce of the mounted zFS aggregate before dumping the data set and an unquiesce when the dump ends. Before performing a backup, review the information in “Unmounting zFS file systems before copying or moving” on page 39 and the following guidelines.

Review the following guidelines before performing a backup of zFS:

1. Do not specify TOL(ENQF) when backing up zFS aggregates because it can cause corruption of the file system.

2. Full volume dumps of volumes that contain mounted zFS file systems will not quiesce the file systems. As a result, all file systems that reside on the volume must be unmounted before performing a full volume dump. For information about logical and full volume dumps, see z/OS DFSMSdfp Storage Administration.

3. The term sysplex as it applies to zFS means a sysplex that supports the z/OS UNIX shared file system environment. That is, a sysplex that has a BPXPRMxx specification of SYSPLEX(YES).

4. If a quiesce is not done before the backup of a mounted file system, corruption of the file system can result. If you are using a different program or different commands than shown in “Backing up a zFS aggregate” on page 64, verify that a quiesce is done (automatically by the backup program) while the back up is occurring. If it is not, then you need to unmount the file system before backing it up or supply a before and after job step to quiesce and then unquiesce the aggregate before and after the backup. The steps are similar to Figure 29.

```plaintext
//*-----------------------------------------------------------------
//* THIS STEP QUIESCES THE AGGREGATE.
//*-----------------------------------------------------------------
QUIESCE EXEC PGM=IOEZADM,REGION=0M,
PARM=('quiesce -aggregate hlq.ZFS.AGGR004')
//* //SYSPRINT DD SYSOUT=H
//STOUT DD SYSOUT=H
//STOERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*
//*-----------------------------------------------------------------
//* THIS STEP UNQUIESCES THE AGGREGATE.
//*-----------------------------------------------------------------
//UQUIESCE EXEC PGM=IOEZADM,REGION=0M,
// PARM=('unquiesce -aggregate hlq.ZFS.AGGR004')
//* //SYSPRINT DD SYSOUT=H
//STOUT DD SYSOUT=H
//STOERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*
```

Figure 29. Steps for quiesce and unquiesce
Backing up a zFS aggregate

![Figure 30](image1) shows an example of a job for backing up a zFS aggregate (and all the file systems). Ensure that the size of the target sequential data set has sufficient space. For additional information about the DUMP command and its keywords, see z/OS DFSMSdfp Storage Administration.

**Important:** Do not specify TOL(ENQF) when backing up zFS aggregates.

```
//ZFSBKUP1 JOB (OS390), 'PROGRAMMER', CLASS=A,
// MSGCLASS=X, MSGLEVEL=(1,1)
//------------------------------------------------------------------
// THIS JOB QUIESCES A ZFS AGGREGATE, DUMPS IT, THEN UNQUIESCES IT.
//------------------------------------------------------------------
//DUMP EXEC PGM=ADRDSSU, REGION=4096K
//SYSPRINT DD SYSOUT=*  
//SYSABEND DD SYSOUT=*  
//OUT DD DSN=hlq.ZFS.AGGR004.BACKUP,  
// DISP=(NEW,CATLG,DELETE), SPACE=(CYL,(5,1),RLSE)  
//SYSIN DD  * DUMP DATASET(INCLUDE(hlq.ZFS.AGGR004))
//  RESET -  
//  OUTDD(OUT)

Figure 30. Job to back up a zFS aggregate
```

Restoring an aggregate with DFSMSdss logical restore

Use DFSMSdss logical restore to restore a zFS aggregate. If the original aggregate (in the example, hlq.ZFS.AGGR004) still exists, the aggregate is restored into a new aggregate (in the example, OMVS.PRIV.AGGR005.LDS0005). ![Figure 31](image2) is an example of a job to restore a zFS aggregate.

```
//ZFSREST1 JOB (OS390), 'PROGRAMMER', CLASS=A,
// MSGCLASS=X, MSGLEVEL=(1,1)
//------------------------------------------------------------------
// THIS JOB RESTORES A ZFS AGGREGATE.
//------------------------------------------------------------------
//ZFSREST EXEC PGM=ADRDSSU, REGION=0M
//SYSPRINT DD SYSOUT=*  
//SYSABEND DD SYSOUT=*  
//INDS DD DISP=SHR, DSN=hlq.AGGR004.BACKUP
//SYSIN DD  *
// RESTORE DATASET(INCLUDE(**)) -  
// CATALOG -  
// RENAMEU( -  
// (hlq.ZFS.AGGR004, -  
// OMVS.PRIV.AGGR005.LDS0005) -  
// ) -  
// WRITECHECK -  
// INDD(INDS)

Figure 31. Job to restore a zFS aggregate
```

For a compatibility mode aggregate, perform the following steps after the aggregate is restored:

1.Unmount the original aggregate (in this case, hlq.ZFS.AGGR004) if it still exists (this also detaches it).
2. Mount the file system in the restored aggregate (in this case, OMVS.PRV.AGGR005.LDS0005).

**Figure 32** is an example of a job to perform a logical restore of a zFS aggregate using DFSMSdss by replacing the existing aggregate. The backup is restored into the original aggregate (in this case, hlq.ZFS.AGGR004). The aggregate cannot be mounted (or attached) during the restore operation.

```
//ZFSREST2 JOB (OS390), 'PROGRAMMER', CLASS=A, // MSGCLASS=X, MSGLEVEL=(1,1) // *--------------------------------------------------------------------------- // ** THIS JOB RESTORES A ZFS AGGREGATE. // *--------------------------------------------------------------------------- //ZFSREST EXEC PGM=ADRDSSU,REGION=0M //SYSPRINT DD SYSOUT=* //SYSABEND DD SYSOUT=* //INDS DD DISP=SHR, DSN=hlq.AGGR004.BACKUP //SYSIN DD * RESTORE DATASET(INCLUDE(hlq.ZFS.AGGR004)) - CATALOG - REPLACE - WRITECHECK - INDD(INDS)
```

**Figure 32. Job to restore a zFS aggregate with replace**

Leading blanks are required before the control statements (RESTORE, CATALOG, RENAMU).

For more information about DFSMSdss logical restore, see [z/OS DFSMSdss Storage Administration](#).

Beginning in z/OS V1R2, zFS enhanced its support for the backup change activity flag in the VTOC (D1DSCHA in the Format 1/8). This flag indicates to a program (like DFSMSHsm) whether the backup of a file system is needed (that is, data in the file system has been modified since the last backup).

In releases prior to z/OS V2R1, zFS would set the change activity flag when a file system was mounted. This is no longer done. Essentially, zFS will cause the setting of the change activity bit in the following cases:

1. During the first write after a MOUNT
2. During the first write after a successful backup (that is, after a successful reset of the change activity flag)
3. During log recovery (that is, during the replay of an aggregate log during the next mount after a system failure)
4. During salvager operation if the log is replayed or a repair is made

The formatting of a new zFS aggregate will not cause the setting of the change activity flag. If an existing zFS aggregate is formatted using the -overwrite option, then the change activity flag will be set.

z/OS V2R1 zFS will also supply an application programming interface, which can be used to reset the change activity flag for a file system. This interface is intended to be used by DFSMSdss when doing a backup of a mounted zFS file system. For more information, see "[Reset Backup Flag](#) on page 271."
Chapter 7. Migrating data from HFS to zFS

This section discusses how to migrate data from HFS to zFS. In all cases, the target file system can be version 1.4 as well as version 1.5. If all of the systems at your site are running at least version z/OS V2R1, then it is recommended that you use version 1.5 aggregates for your new file systems.

Guideline: Do not use the HFS to zFS migration tool if you are migrating your syplex root. To migrate the syplex root, consider using the MODIFY OMVS,NEWROOT operator command. For details, see the topic Steps for dynamically replacing the syplex root file system in z/OS UNIX System Services Planning.

Using the z/OS HFS to zFS migration tool

You can use the ISPF-based BPXWH2Z tool to migrate HFS file systems to zFS file systems. It has a panel interface that enables you to alter the space allocation, placement, SMS classes and data set names. With this tool, you can:

- Migrate HFS file systems (both mounted and unmounted) to zFS file systems. If the HFS being migrated is mounted, the tool automatically unmounts it and then mounts the new zFS file system on its current mount point.
- Define zFS aggregates, using the default settings, to be approximately the same size as the HFS. The new allocation size can also be increased or decreased.
- Have the migration run in TSO foreground or z/OS UNIX background.

The number of storage blocks needed to store a zFS file system might not be exactly the same as the amount needed for HFS. For example, starting with z/OS V1R13, zFS uses 8K blocks to contain small files; however, HFS uses 4K blocks. In this case, some HFS file systems may need additional storage (possibly twice as much) when they are migrated to zFS. For more information about migrating data from HFS to zFS, see z/OS Migration.

Note that when BPXWH2Z creates new zFS aggregates, you can control whether it creates version 1.4 aggregates or version 1.5 aggregates. The default is to create version 1.4 aggregates. You can change this default by specifying format_aggrversion=5 in your IOEPRMxx configuration options file before IPL or by dynamically changing the option by using the zfsadm config -format_aggrversion 5 command.

Using the z/OS UNIX pax command

You can copy data from an HFS file system to a zFS file system by using the z/OS UNIX pax command with or without using an intermediate archive file. See z/OS UNIX System Services Command Reference for more information about the pax command. When the data is being copied, the file system being accessed must be mounted.

Note: If you are migrating a file system that contains additional file systems mounted below it, the default settings on the pax command also copies the files and directories contained in those file systems. To avoid this, you can either specify the pax -X option, or unmount the lower file systems before issuing the pax command.
Using an intermediate archive file

Use the `pax` command to copy the source (HFS) file system into an intermediate archive file and then use the `pax` command to copy from the archive file into the target (zFS) file system. This archive file can be a z/OS UNIX file or it can be an MVS™ data set.

Suppose you have an HFS file system mounted at `/etc/dfs`. You want to copy this into an empty zFS file system mounted at `/etc/dir1/testzfs1`. You issue the following commands from z/OS UNIX:

1. Move to the source (HFS) file system mounted at `/etc/dfs`
   ```bash
cd /etc/dfs
   ```
2. Create a z/OS UNIX archive file called `/tmp/zfs1.pax` that contains the HFS file system mounted at `/etc/dfs`
   ```bash
   pax -wvf /tmp/zfs1.pax .
   ```
3. Move to the target (zFS) file system mounted at `/etc/dir1/testzfs1`
   ```bash
cd /etc/dir1/testzfs1
   ```
4. Read the archive file into the zFS file system mounted at `/etc/dir1/testzfs1`
   ```bash
   pax -rv -p e -f /tmp/zfs1.pax
   ```

Without using an intermediate archive file

Use the `pax` command to copy the source (HFS) file system to the target (zFS) file system, without an intermediate archive file.

Suppose you have an HFS file system mounted at `/etc/dfs`. You want to copy this into an empty zFS file system mounted at `/etc/dir1/testzfs1`. You issue the following commands from OMVS:

1. Move to the source (HFS) file system mounted at `/etc/dfs`
   ```bash
cd /etc/dfs
   ```
2. Copy the (HFS) file system mounted at `/etc/dfs` to the (zFS) file system mounted at `/etc/dir1/testzfs1`
   ```bash
   pax -rwCMX -p eW . /etc/dir1/testzfs1
   ```
Chapter 8. Performance and debugging

This section discusses performance tuning techniques and what should be done if a problem occurs that requires IBM service assistance. The examples presented here are for illustrative purposes only; it is typical for the output of some reports to wrap.

Performance tuning

zFS performance is dependent on many factors. zFS provides performance information to help the administrator determine bottlenecks. The IOEFSPRM file contains many tuning options that can be adjusted. The output of the system modify zfs, query commands provide feedback about the operation of zFS. This section describes those IOEFSPRM options and the operator commands that relate to performance.

It is always better for performance in a shared file system environment if you can mount a file system read-only rather than read/write. For example, the sysplex root file system and the version file systems perform better if they are mounted read-only. For more information about sharing file systems in a sysplex, see z/OS UNIX System Services Planning.

In addition, if a file system is mounted read/write, but accessed mainly from one system (for instance, SY1), it is better for performance if that file system is z/OS UNIX owned on that system (SY1). To keep z/OS UNIX ownership on SY1, you might want to mount it with the UNMOUNT option or the NOAUTOMOVE option. If you must use the AUTOMOVE option because you want the file system to remain available even when SY1 is down, move z/OS UNIX ownership of that file system back to SY1 when SY1 becomes available. This is not necessary for zFS read/write file systems that are sysplex-aware.

zFS performance can be optimized by tailoring the size of its caches to reduce I/O rates and pathlength. It is also important to monitor DASD performance to ensure that there are no volumes or channels that are pushed beyond their capacity. The following sections describe areas to consider when tuning zFS performance.

Total cache size

The total storage size for all of the caches in the zFS address space must be less than 2 GB. zFS terminates when it cannot obtain all the storage it needs for the caches specified in IOEFSPRM file. In addition to the zFS address space caches, storage is necessary for processing file requests and for the products zFS might use. As a result, you must restrict the total zFS address space cache storage to approximately 1.5 GB. Use modify zfs,query,storage to determine the total allocated zFS storage. See STOR” on page 89 for more information about determining how much of the available zFS address space storage is being used by the zFS caches.

You might see messages IOEZ00188A, IOEZ00662I, or IOEZ00663I, which indicate that the zFS address space is low on storage. If it continues to be a problem, consider using zfsadm config to dynamically decrease the metadata, transaction, and vnode cache sizes.
The `modify zfs,query,all` command also shows the total zFS storage allocated, but includes the storage allocated for all the caches and everything else zFS might need. The zFS address space caches include:

- "Metadata cache"
- "Transaction cache" on page 71
- "Vnode cache" on page 71

The user data cache, client cache, log file cache, and metadata backing cache are stored in data spaces and do not use zFS address space storage.

**Metadata cache**

The metadata cache is used to contain all file system metadata; this metadata includes all directory contents, file status information (such as, atime, mtime, size, and permission bits), and file system structures. Additionally, it also caches data for files smaller than 7 K. Essentially, zFS stores a file by using one of the following three methods. For more information about how zFS shows free blocks, see "Understanding zFS disk space allocation" on page 40.

**inline** If the file is 52 bytes or less, its data is stored in the structure that contains the status information for the file.

**fragmented**

Beginning with z/OS V1R13, new zFS user data is not stored in fragments. Files that are 53 bytes or more are stored as blocked. The first time a fragmented file is updated, it is converted to blocked. On prior releases, if the file is less than 7 K, it is stored in blocks on disk that can be shared with other files; multiple files are then stored in the same physical disk block. Physical disk blocks are always 8 Ki in size.

**blocked**

Beginning with z/OS V1R13, new zFS user data is not stored in 1 K fragments. Files that are 53 bytes or more are stored as blocked. The first time a fragmented file is updated, it is converted to blocked. On prior releases, files larger than 7 K are stored in multiple blocks, blocked files are only stored in the user file cache, and all I/O is performed directly to or from user file cache buffers.

Because inline files are stored in the status block, files that are stored on disk by using the inline method are stored in the metadata and hence are cached in the metadata cache (and also in the user file cache). Because the metadata cache is the only component that knows about multiple files sharing disk blocks, small fragmented files are stored in the metadata cache (and also in the user file cache) and I/O is performed directly to or from the metadata cache for these small user files.

Generally, metadata is referred to and updated frequently for most zFS file operations; hence, achieving a good hit ratio is often essential to good performance for most workloads. A good hit ratio might be considered to be 90% or more, depending on your workload.

The metadata cache is stored in the primary address space and its default size is 64 M. Because the metadata cache contains only metadata and small files, it typically does not need to be nearly as large as the user file cache. The operator `modify zfs,query,all` command output shows statistics for the metadata cache including the cache hit ratio.
An optional metadata backing cache can be specified that extends the size of the metadata cache. It resides in a data space and increases the amount of metadata that can be kept in memory. It might improve the performance of workloads that require large amounts of metadata.

**Note:** Beginning with z/OS V1R13, the optional metadata backing cache cannot be created dynamically using the `zfsadm config -metaback_cache_size` command. You must specify the `metaback_cache_size` configuration option in your IOEFSPRM file to have a metadata backing cache. You can still use the `zfsadm config` command to dynamically change the size of the metadata backing cache when the metadata backing cache exists.

### Transaction cache

Every change to zFS file system metadata is bounded by a transaction that describes its changes by using records written to the log file. The transaction cache is a cache of data structures representing transactions.

The transaction cache is stored in the zFS primary address space with a default of 2000 transactions. zFS dynamically increases the size of this cache based on the number of concurrent pending transactions (transactions that have not been fully committed to disk) in the zFS file system. Therefore, the administrator does not need to tailor the transaction cache size. However, the `modify zfs,query,all` output shows the transaction count at any given time.

### Vnode cache

Every object in the zFS file system is represented by a data structure called a vnode in memory. zFS keeps a cache of these vnodes and recycles them in a least recently used (LRU) manner. Every operation in zFS requires a vnode and z/OS UNIX keeps pointers to zFS vnodes. Because z/OS UNIX keeps references to zFS vnodes, zFS might be forced to dynamically increase the size of this cache to meet the demands of z/OS UNIX. To create a zFS vnode for a newly referenced file or a newly created file for a user requires the pathlength to initialize the structure and obtain its status information from the metadata cache. If the status of the file is not in the metadata cache, then a disk I/O might also be required.

The vnode cache is stored in the zFS primary address space and the default number of vnodes is 32,768. As with any cache, a good hit ratio is desirable and the operator `modify zfs,query,all` command shows the vnode cache hit ratio. Because the vnode cache is backed by the metadata cache, if the vnode hit ratio is low but the metadata cache hit ratio is high your performance might not suffer too much because a vnode cache miss requires only some pathlength to initialize the vnode structures.

### User file and client file cache

The user file cache is used to cache all “regular” files. It caches any file, no matter what its size, and performs write-behind and asynchronous read-ahead for files. It performs I/O for all files that are 7 KB or larger. For files smaller than 7 KB, I/O is typically performed through the metadata cache. When you use sysplex-aware file systems, zFS also has a client cache that is used when systems running z/OS releases before V1R13 are present in the shared file system environment.

The user file and client cache are allocated in data spaces. The default sizes are 256 MB for the `user_cache_size` and 32 MB for the `client_cache_size` (this option is only meaningful in mixed release environments with releases before z/OS V1R13).
However, you can tailor these sizes to meet your performance needs, based on your overall system memory. The maximum size for both the user_cache_size and client_cache_size is 65,536 MB (64 GB). The general rule for any cache is to ensure a good hit ratio. Additionally, it is good to have a user file cache that is large enough for write-behind activity to occur. If the cache is too small, you need to recycle buffers more frequently and that might degrade write-behind performance. The `modify zfs,query,all` command output shows the cache hit ratio, which is actually the “fault ratio”. To get the hit ratio, subtract the fault ratio from 100%.

In general, you should have hit ratios of at least 80% or more. Hit ratios over 90% typically give good performance. However, the hit ratio is very much workload-dependent. For example, a zFS file system that is exported exclusively to SMB clients by using the SMB server will likely have a low hit ratio. The low hit ratio occurs because the SMB client and the SMB server cache data, which reduces the zFS cache hit ratio. This reduction is expected and is not considered a problem.

Log files

Every zFS aggregate contains a log file that is used to record transactions that describe changes to the file system structure. This log file is, by default, 1% of the aggregate size; but, you can tailor it on the `i o e a g f m t` command. Typically, 1% is sufficient for most aggregates. However, larger aggregates might need less than 1%, while very small aggregates might need more than 1% if a high degree of parallel update activity occurs for the aggregate.

Log file cache

The log file cache is a pool of 8 KB buffers used to contain log file updates. You must not modify the log file cache size unless under the direction of IBM service. Log file buffers are always written asynchronously to disk and typically need to be waited upon only when the log is becoming full, or if a file is in file synchronization (fsync).

The log file cache is stored in a data space and its default size is 16 MB. The log file cache is grown dynamically by adding one 8 KB buffer for each attached aggregate. This growth ensures that each aggregate always has one 8 KB buffer to use to record its most recent changes to file system metadata. Because log files are written asynchronously, the cache essentially allows write-behind of log files and because the cache is shared among all aggregates. Aggregates that have a higher write rate use more buffers in the cache using a least-recently-used (LRU) algorithm.

The log file cache is a write-only cache. The read hit ratio is not important. Make the log file cache large enough to avoid frequent log file I/O waits. The `modify zfs,query,all` command shows log file I/O waits. However, every workload is different. For example, workloads that issued fsync operations force zFS to sync the log file more frequently.

Fixed storage

By default, zFS does not fix pages in any of the caches except when an I/O is pending to or from the cache buffers. The administrator can permanently page fix the user file cache, the metadata cache, metaback_cache_size, and the log file cache by choosing the fixed option for the cache. This option ensures that the cache experiences no paging and avoids page fixing for each I/O. This option does come at the expense of using real storage for the cache, which means the real storage is not available for other applications.
If your file system performance is critical and you have enough real memory to support it, the fixed option can be useful. Otherwise, do not set it.

**I/O balancing**

The performance of any file system is heavily dependent on DASD I/O performance. If any channels or DASD volumes are overloaded, then it is possible for excessive I/O waits to occur on that DASD. Performance products such as RMF™ show DASD performance.

zFS modify zfs,query,all operator commands also provide reports that show I/O rates per aggregate, and file system request rates per aggregate and per file system. This information, along with DASD performance information from RMF or performance products similar to RMF can be used to balance I/O among your DASD. For example, you can use the query command output to show the file systems that can be moved to different DASD to achieve a better balance among disks.

**Monitoring zFS performance**

You can monitor zFS performance using the MODIFY command. The output from the MODIFY ZFS,QUERY command is written to the system log. The syntax of this command and an explanation of the report and their option values, if any, are shown below.

```
modify zfs,query,<report>,<option>
```

**KN** Provides counts of calls made to zFS from z/OS UNIX and the average response time of each call. This information is the basic measure of zFS performance. See “KN” on page 78 for details of the report.

**VM** Provides performance information for the user file cache including cache hit ratios, I/O rates, and storage usage. See “VM” on page 94 for details of the report.

**LFS** Provides detailed file system statistics including the performance of the zFS metadata caches, the vnode cache, and the aggregate I/O statistics. See “LFS” on page 79 for details of the report.

**LOG** Provides performance information for the log file cache. See “LOG” on page 88 for details of the report.

**LOCK** Provides a measure of lock contention and how often z/OS UNIX threads wait for certain events such as user file cache reclaim. See “LOCK” on page 85 for details of the report.

**STOR** Provides a detailed breakdown of zFS allocated storage by component. By default, this report lists only storage usage by zFS component. If you use the DETAILS option, you get more detailed information for each zFS component. See “STOR” on page 89 for details of the report.

**FILE** Provides a detailed breakdown of requests per zFS file system and aggregate. By default, this report lists only file systems and aggregates that had active requests since the last statistics reset. If you use the ALL option, you get all file system and aggregates regardless of whether they were active or not. See “FILE” on page 76 for details of the report.
STKM
Displays the current server token manager (STKM) statistics. See “STKM” on page 88 for details of the report.

CTKC
Displays the client token manager statistics. CTKC is only present when the system is a Sysplex client of another system and the zFS ctkc component on this system sent a message to another system. See “CTKC” on page 75 for details of the report.

IOBYDASD
Displays the I/O statistics by currently attached DASD volumes including the total number of waits for I/O and the average wait time per I/O. See “IOBYDASD” on page 77 for details of the report.

DATASET
Displays zFS statistics about file systems.

SVI
Displays the server vnode interface (SVI) calls from other systems to this server. See “SVI” on page 93 for details of the report. SVI will be present when the system is a server to another system and the zFS svi component on this system has received a message from another system.

ALL
Shows all the above reports. However, for the STOR report, the DETAILS option is off and the FILE report indicates only active file systems.

Resetting performance monitoring data
You can reset the performance monitoring statistics for any given zFS report or reset all of the internal zFS statistics. The syntax of this command is shown below, where report is KN, VM, LFS, LOG, LOCK, STOR, FILE, STKM, CTKC, IOBYDASD, DATASET, SVI, or ALL.

```
modify zfs,reset,<report>
```

Resetting the statistics is useful if you want to view zFS performance for a given time of day, such as during peak usage. For example, if you want performance of zFS between 1 PM and 3 PM, you enter MODIFY ZFS,RESET,ALL at 1 PM and enter MODIFY ZFS,QUERY,ALL at 3 PM.

To start the monitoring period at 1 PM, enter modify zfs,reset,all.
To end the monitoring period at 3 PM, enter modify zfs,query,all.

Sample zFS QUERY reports
The following sections show sample output from zFS QUERY reports and describe the relevant fields of each report. Some fields are used mainly by IBM service, but are included here for completeness.

- “CTKC” on page 75
- “DATASET” on page 75
- “FILE” on page 76
- “IOBYDASD” on page 77
- “KN” on page 78
- “LFS” on page 79
- “LOCK” on page 85
- “STKM” on page 88
- “STOR” on page 89
- “SVI” on page 93
- “VM” on page 94
CTKC

Figure 33 shows an example of the total number of call counts and the average response time in milliseconds of the call to the system indicated (in this case NP1).

**Note:** Output is only displayed when the zFS ctkc component on this system has sent a message to another system.

<table>
<thead>
<tr>
<th>SVI Call</th>
<th>Count</th>
<th>Avg. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetToken</td>
<td>211324</td>
<td>15.996</td>
</tr>
<tr>
<td>GetMultTokens</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>ReturnTokens</td>
<td>31</td>
<td>0.621</td>
</tr>
<tr>
<td>ReturnFileTokens</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FetchData</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>StoreData</td>
<td>27005</td>
<td>3.354</td>
</tr>
<tr>
<td>Setattr</td>
<td>184762</td>
<td>4.486</td>
</tr>
<tr>
<td>FetchDir</td>
<td>25</td>
<td>20.464</td>
</tr>
<tr>
<td>Lookup</td>
<td>30</td>
<td>4.772</td>
</tr>
<tr>
<td>GetTokensDirSearch</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>Create</td>
<td>3</td>
<td>17.921</td>
</tr>
<tr>
<td>Remove</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>Rename</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>Link</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>ReadLink</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>SetACL</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>Statfs</td>
<td>42</td>
<td>2.006</td>
</tr>
<tr>
<td>TSR</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FileSysSyncTable</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FileSyncMeta</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>BitmapReserve</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>BitmapUnreserve</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>BitmapReclaim</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FileUpdateIB</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FileCreateIB</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FwdReadDir</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>LkupInvalidate</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FileDebug</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FetchPage</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>ServerIO</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>BulkFetchStatus</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>Convert</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>ConvertFID</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td><strong>TOTALS</strong></td>
<td>423222</td>
<td>10.162</td>
</tr>
</tbody>
</table>

Figure 33. Sample CTKC report

DATASET

The DATASET report, shown in Figure 34 on page 76 lists zFS data set statistics. Table 2 on page 76 describes the contents of the report.
Table 2. DATASET report fields

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocates</td>
<td>Number of allocations issued by zFS for zFS data sets.</td>
</tr>
<tr>
<td>Allocates failed</td>
<td>Number of allocations issued by zFS for zFS data sets that were unsuccessful.</td>
</tr>
<tr>
<td>Unallocates</td>
<td>Number of unallocations issued by zFS for zFS data sets.</td>
</tr>
<tr>
<td>Unallocates failed</td>
<td>Number of unallocations issued by zFS for zFS data sets that were unsuccessful.</td>
</tr>
<tr>
<td>Opens</td>
<td>Number of opens issued by zFS for zFS data sets.</td>
</tr>
<tr>
<td>Opens failed</td>
<td>Number of opens issued by zFS for zFS data sets that were unsuccessful.</td>
</tr>
<tr>
<td>Closes</td>
<td>Number of closes issued by zFS for zFS data sets.</td>
</tr>
</tbody>
</table>

**FILE**

The FILE report, which is shown in Figure 35, lists every file system that was active since the last reset by default. If you use the **ALL** option, it lists all file systems. The file systems are listed in the report with the most active file systems listed first. Table 3 describes the contents of the report.

Table 3. FILE report fields

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggr #</td>
<td>The aggregate ID that can be seen in the <code>zfsadm lsf -long</code> command.</td>
</tr>
</tbody>
</table>

---

Figure 34. Sample DATASET report

Table 2. DATASET report fields

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocates</td>
<td>Number of allocations issued by zFS for zFS data sets.</td>
</tr>
<tr>
<td>Allocates failed</td>
<td>Number of allocations issued by zFS for zFS data sets that were unsuccessful.</td>
</tr>
<tr>
<td>Unallocates</td>
<td>Number of unallocations issued by zFS for zFS data sets.</td>
</tr>
<tr>
<td>Unallocates failed</td>
<td>Number of unallocations issued by zFS for zFS data sets that were unsuccessful.</td>
</tr>
<tr>
<td>Opens</td>
<td>Number of opens issued by zFS for zFS data sets.</td>
</tr>
<tr>
<td>Opens failed</td>
<td>Number of opens issued by zFS for zFS data sets that were unsuccessful.</td>
</tr>
<tr>
<td>Closes</td>
<td>Number of closes issued by zFS for zFS data sets.</td>
</tr>
</tbody>
</table>

---

Figure 35. Sample FILE report

Table 3. FILE report fields

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggr #</td>
<td>The aggregate ID that can be seen in the <code>zfsadm lsf -long</code> command.</td>
</tr>
</tbody>
</table>
Table 3. FILE report fields (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flg</td>
<td>Indicates the aggregate status, as follows:</td>
</tr>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>G</td>
</tr>
<tr>
<td></td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>Q</td>
</tr>
<tr>
<td></td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>This command only reports on locally mounted (attached) aggregates. You can use the operator ROUTE command to issue this command to all systems in your sysplex (for example, ROUTE *ALL,F ZFS,QUERY,FILE,ALL). Note that the zFS owning system can flag an aggregate as growing (G) while the other (zFS client) systems can flag it as quiesced (Q). This is because an aggregate that is growing is quiesced on all other systems.</td>
</tr>
<tr>
<td>Operations</td>
<td>Indicates the count of z/OS UNIX vnode calls to that particular file system; it is not an I/O rate. You can use the RMF DASD reports, the LFS Aggregate I/O report, and the FILE report to balance your file systems among disks to provide a more even I/O spread.</td>
</tr>
</tbody>
</table>

IOBYDASD

As Figure 36 shows, the IOBYDASD report lists the currently attached DASD by volume. This report is important for viewing the average wait time per I/O (in milliseconds).

Table 4 describes the contents of the report.

Table 4. IOBYDASD report fields

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DASD VOLSER</td>
<td>The DASD volumes that contain the zFS aggregates.</td>
</tr>
<tr>
<td>PAV IOs</td>
<td>The maximum number of concurrent I/O requests to volume.</td>
</tr>
<tr>
<td>Reads</td>
<td>The number of read I/O requests.</td>
</tr>
<tr>
<td>K bytes</td>
<td>The number of bytes read or written in K units.</td>
</tr>
<tr>
<td>Writes</td>
<td>The number of write I/O requests.</td>
</tr>
<tr>
<td>Waits</td>
<td>The number of waits for I/O completion.</td>
</tr>
<tr>
<td>Average Wait</td>
<td>The average wait time for I/O requests in milliseconds.</td>
</tr>
</tbody>
</table>

| Total number of waits for I/O | Total of Waits column |
Table 4. IOBYDASD report fields (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average wait time per I/O</td>
<td>The average of the Average Wait times, in milliseconds.</td>
</tr>
</tbody>
</table>

**KN**

The QUERY,KN report shows basic zFS performance for both the PFS file system owner and the PFS client. It shows all calls made to zFS by z/OS UNIX since the last statistics reset or since zFS was first initialized if no explicit reset has been done, and the average response time in milliseconds for each request. These requests are the official interface between z/OS UNIX and zFS; this is the most fundamental measure of zFS performance because it includes any CPU, I/O wait time, or lock wait time.

The times here represent only the zFS portion of the overall command response time. For example, entering a `mkdir` command from z/OS UNIX will actually result in many zFS calls, and the `zfs_mkdir` time is only the portion of time it took zFS to perform the actual `mkdir`. Hence, application time and time spent processing in z/OS UNIX is not included here.

If you see abnormally long times that are listed for `zfs_lookup`, `zfs_creates`, or `zfs_removes` and you are using v4 directories, you might have a ZFS large directory problem. For information about the ZFS large directory performance problem, see "Minimum and maximum file system sizes" on page 44.

In the following sample KN report, the **Operation** column is the z/OS UNIX operation being performed, the **Count** column is the number of operations, the **XCF Reqn** column is the number of XCF messages that were sent during the processing of the operation and **Avg Time** is the average response time for the operations. The server could send XCF messages to revoke tokens and the client might send XCF messages to obtain needed tokens and security information from a server or to write metadata changes to the server. If XCF messages need to be sent, then you should expect average response times to be longer than if messages were not sent.

```
F ZFS,QUERY,KNPFS
IOEZ00438I Starting Query Command KN. 525
PFS Calls on Owner

<table>
<thead>
<tr>
<th>Operation</th>
<th>Count</th>
<th>XCF req.</th>
<th>Avg Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>zfs_opens</td>
<td>9456</td>
<td>0</td>
<td>1.778</td>
</tr>
<tr>
<td>zfs_closes</td>
<td>9444</td>
<td>2</td>
<td>0.788</td>
</tr>
<tr>
<td>zfs_reads</td>
<td>12363</td>
<td>0</td>
<td>0.145</td>
</tr>
<tr>
<td>zfs_writes</td>
<td>109510</td>
<td>0</td>
<td>2.475</td>
</tr>
<tr>
<td>zfs_ioctl</td>
<td>66</td>
<td>0</td>
<td>0.685</td>
</tr>
<tr>
<td>zfs_getattts</td>
<td>17561</td>
<td>1</td>
<td>0.033</td>
</tr>
<tr>
<td>zfs_setattts</td>
<td>699</td>
<td>0</td>
<td>0.377</td>
</tr>
<tr>
<td>zfs_accesses</td>
<td>709</td>
<td>0</td>
<td>0.099</td>
</tr>
<tr>
<td>zfs_lookups</td>
<td>52931</td>
<td>959</td>
<td>0.396</td>
</tr>
<tr>
<td>zfsCreates</td>
<td>3158</td>
<td>780</td>
<td>4.659</td>
</tr>
<tr>
<td>zfs_removes</td>
<td>633</td>
<td>0</td>
<td>4.164</td>
</tr>
<tr>
<td>zfs_links</td>
<td>16</td>
<td>0</td>
<td>0.978</td>
</tr>
<tr>
<td>zfs_renames</td>
<td>640</td>
<td>0</td>
<td>2.108</td>
</tr>
<tr>
<td>zfs_mkdirs</td>
<td>780</td>
<td>164</td>
<td>4.346</td>
</tr>
<tr>
<td>zfs_rmdirs</td>
<td>378</td>
<td>0</td>
<td>4.731</td>
</tr>
<tr>
<td>zfs_readdirs</td>
<td>440</td>
<td>23</td>
<td>5.871</td>
</tr>
<tr>
<td>zfs_smlinks</td>
<td>59</td>
<td>19</td>
<td>4.393</td>
</tr>
<tr>
<td>zfs_readlinks</td>
<td>132</td>
<td>0</td>
<td>0.076</td>
</tr>
</tbody>
</table>
```
### zfs Fsyncs
- zfs_fsyncs: 188
- zfs_truncs: 0
- zfs_lockctls: 0
- zfs_audits: 0
- zfs_inactives: 0
- zfs_recoveries: 0
- zfs_lockctls: 0
- zfs_statfss: 245
- zfs_mounts: 22
- zfsUnmounts: 0
- zfs_vinacts: 0

*TOTALS* 334045

**IOEZ00438I** Starting Query Command KN. 526

**PFS Calls on Client**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Count</th>
<th>XCF req.</th>
<th>Avg Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>zfs_opens</td>
<td>6373</td>
<td>121</td>
<td>2.666</td>
</tr>
<tr>
<td>zfs_closes</td>
<td>6365</td>
<td>2675</td>
<td>2.243</td>
</tr>
<tr>
<td>zfs_reads</td>
<td>17402</td>
<td>2013</td>
<td>0.636</td>
</tr>
<tr>
<td>zfs_writes</td>
<td>12043</td>
<td>8904</td>
<td>10.122</td>
</tr>
<tr>
<td>zfs_ioctl</td>
<td>15</td>
<td>0</td>
<td>0.096</td>
</tr>
<tr>
<td>zfs_getattr</td>
<td>14560</td>
<td>3972</td>
<td>1.170</td>
</tr>
<tr>
<td>zfs_setattr</td>
<td>1183</td>
<td>17</td>
<td>0.102</td>
</tr>
<tr>
<td>zfs_accesses</td>
<td>194</td>
<td>0</td>
<td>0.331</td>
</tr>
<tr>
<td>zfs_lookup</td>
<td>47122</td>
<td>1606</td>
<td>0.340</td>
</tr>
<tr>
<td>zfs_create</td>
<td>2492</td>
<td>2491</td>
<td>8.749</td>
</tr>
<tr>
<td>zfs_removes</td>
<td>485</td>
<td>485</td>
<td>6.049</td>
</tr>
<tr>
<td>zfs_links</td>
<td>15</td>
<td>15</td>
<td>4.582</td>
</tr>
<tr>
<td>zfs_rename</td>
<td>1151</td>
<td>1151</td>
<td>5.673</td>
</tr>
<tr>
<td>zfs_mkdir</td>
<td>603</td>
<td>603</td>
<td>6.548</td>
</tr>
<tr>
<td>zfs_rmdir</td>
<td>582</td>
<td>582</td>
<td>5.834</td>
</tr>
<tr>
<td>zfs_readdir</td>
<td>317</td>
<td>27</td>
<td>3.625</td>
</tr>
<tr>
<td>zfs_smlink</td>
<td>55</td>
<td>55</td>
<td>0.031</td>
</tr>
<tr>
<td>zfs_readlink</td>
<td>115</td>
<td>72</td>
<td>1.854</td>
</tr>
<tr>
<td>zfs_fsyncs</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>zfs_truncs</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>zfs_lockctls</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>zfs_audits</td>
<td>92</td>
<td>0</td>
<td>0.021</td>
</tr>
<tr>
<td>zfs_inactives</td>
<td>2077</td>
<td>0</td>
<td>0.016</td>
</tr>
<tr>
<td>zfs_recoveries</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>zfs_vgets</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>zfs_pfsctls</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>zfs_statfss</td>
<td>50</td>
<td>50</td>
<td>4.741</td>
</tr>
<tr>
<td>zfs_mounts</td>
<td>7</td>
<td>0</td>
<td>313.115</td>
</tr>
<tr>
<td>zfsUnmounts</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>zfs_vinacts</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
</tbody>
</table>

*TOTALS* 113298

**IOEZ000251** zFS kernel: MODIFY command - QUERY,KNPFS completed successfully.

### LFS

The LFS report provides detailed file system statistics; the following sample shows an example of the content. Each part of the report is described in the following sample LFS report.

**F ZFS,QUERY,LFS**

**IOEZ00438I** Starting Query Command LFS. 421

**zfs Vnode Op Counts**

<table>
<thead>
<tr>
<th>Vnode Op</th>
<th>Count</th>
<th>Vnode Op</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>efs_hold</td>
<td>0</td>
<td>efs_readdir</td>
<td>67997</td>
</tr>
<tr>
<td>efs_rele</td>
<td>0</td>
<td>efs_create</td>
<td>1569039</td>
</tr>
<tr>
<td>efs_inactive</td>
<td>0</td>
<td>efs_remove</td>
<td>1945874</td>
</tr>
</tbody>
</table>

Chapter 8. Performance and debugging  79
efsvn_getattr 9856523  efs_rename 235320
efs_setattr 40  efs_mkdir 237359
efs_lookup 21545682  efs_link 237318
efs_getvolume 0  efs_rmdir 238004
efs_getlength 0  efs_readlink 0
efs_atfsid 0  efs_fsid 0
efs_cname 0  efs_fsync 0
efs_vname 0  efs_waitio 9
efs_uwrite 0  efs_ccancel 0
efs_uri 0  efs_fsync 0
efs_getnode 16640  efs_vmlkinfo 0
efs_readi 0  efs_convi 0
Average number of names per convert 0
Number of version5 directory splits 126
Number of version5 directory merges 63
Total zFS Vnode Ops 37849050

zFS Vnode Cache Statistics

<table>
<thead>
<tr>
<th>Vnodes</th>
<th>Requests</th>
<th>Hits</th>
<th>Ratio</th>
<th>Allocates</th>
<th>Deletes</th>
</tr>
</thead>
<tbody>
<tr>
<td>200000</td>
<td>25908218</td>
<td>22431383</td>
<td>86.580%</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

zFS Vnode structure size: 224 bytes
zFS extended vnodes: 200000, extension size 788 bytes (minimum)
Held zFS vnodes: 2914 (high 29002) Open zFS vnodes: 0 (high 10) Reusable: 197085
Total osi_getvnode Calls: 3886774 (high resp 0) Avg. Call Time: 0.069 (msecs)
Total SAF Calls: 11050540 (high resp 1) Avg. Call Time: 0.008 (msecs)

zFS Fast Lookup Statistics

<table>
<thead>
<tr>
<th>Buffers</th>
<th>Lookups</th>
<th>Hits</th>
<th>Ratio</th>
<th>Neg. Hits</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0</td>
<td>0</td>
<td>0.0%</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Metadata Caching Statistics

<table>
<thead>
<tr>
<th>Buffers (K bytes)</th>
<th>Requests</th>
<th>Hits</th>
<th>Ratio</th>
<th>Updates</th>
<th>PartialWrt</th>
</tr>
</thead>
<tbody>
<tr>
<td>32768</td>
<td>262144</td>
<td>77813570</td>
<td>99.6%</td>
<td>27943073</td>
<td>423524</td>
</tr>
</tbody>
</table>

Metadata Backing Caching Statistics

<table>
<thead>
<tr>
<th>Buffers (K bytes)</th>
<th>Requests</th>
<th>Hits</th>
<th>Ratio</th>
<th>Discards</th>
</tr>
</thead>
<tbody>
<tr>
<td>131072</td>
<td>1048576</td>
<td>24303</td>
<td>377</td>
<td>1.5%</td>
</tr>
</tbody>
</table>

Transaction Cache Statistics

Transactions started: 7152165 Lookups on tran: 8032713 EC Merges: 516363
Allocated Transactions: 8034 (Act= 0, Pend= 0, Comp= 4846, Free= 3188)

I/O Summary By Type

<table>
<thead>
<tr>
<th>Count</th>
<th>Waits</th>
<th>Cancels</th>
<th>Merges</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>33006</td>
<td>7701</td>
<td>0</td>
<td>0</td>
<td>File System Metadata</td>
</tr>
<tr>
<td>680516</td>
<td>1020</td>
<td>0</td>
<td>56366</td>
<td>Log File</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>User File Data</td>
</tr>
</tbody>
</table>

I/O Summary By Circumstance

<table>
<thead>
<tr>
<th>Count</th>
<th>Waits</th>
<th>Cancels</th>
<th>Merges</th>
<th>Circumstance</th>
</tr>
</thead>
<tbody>
<tr>
<td>7213</td>
<td>6553</td>
<td>0</td>
<td>0</td>
<td>Metadata cache read</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>User file cache direct read</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>Log file read</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Metadata cache async delete write</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Metadata cache async write</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Metadata cache lazy write</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Metadata cache sync delete write</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Metadata cache sync write</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>User file cache direct write</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Metadata cache file sync write</td>
</tr>
<tr>
<td>16991</td>
<td>861</td>
<td>0</td>
<td>0</td>
<td>Metadata cache sync daemon write</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Metadata cache aggregate detach write</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Metadata cache buffer block reclaim write</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Metadata cache buffer allocation write</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Metadata cache file system quiesce write</td>
</tr>
<tr>
<td>8811</td>
<td>286</td>
<td>0</td>
<td>0</td>
<td>Metadata cache log file full write</td>
</tr>
</tbody>
</table>
Table 5. LFS report sections

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>zFS Vnode Op Counts:</td>
<td>Shows the number of calls to the lower layer zFS components. One request from z/OS UNIX typically requires more than one lower-layer call. Note that the output of this report wraps.</td>
</tr>
<tr>
<td>zFS Vnode Cache Statistics:</td>
<td>Shows the zFS vnode cache statistics. It shows the number of currently allocated vnodes and the vnode hit ratio. Allocates and <em>Deletes</em> show requests to create new vnodes (for operations such as create or mkdir) and delete vnodes (for operations such as remove or failed creates or mkdirs). The size of this cache is controlled by the vnode_cache_size parameter and the demand for zFS vnodes placed by z/OS UNIX. In general, zFS tries to honor the setting of the vnode_cache_size parameter and recycle vnode structures to represent different files. However, if z/OS UNIX requests more vnodes than zFS has allocated then zFS must allocate vnodes to avoid applications failing. <strong>Held zFS vnodes</strong> is the number of vnodes that z/OS UNIX has required of zFS to currently access. <strong>high</strong> is the largest number of vnodes that z/OS UNIX required of zFS to access at one time (during a peak time). z/OS UNIX also determines when files are to be opened and closed. <strong>Open zFS vnodes</strong> is the number of vnodes that represent currently open files. <strong>high</strong> is the largest number of files open at the same time. Generally, a good hit ratio for this cache is preferable because a miss means initializing the data structures and initialization requires a read of the object's status from disk. Often this is in the metadata cache, but it is not guaranteed. Consequently a vnode cache lookup miss might require an I/O wait. The vnode structure size is shown; however, additional data structures anchored from the vnode also take space. Everything added together yields over 1 K of storage per vnode. Consider this when planning the size of this cache. Also note that initializing a vnode will not require an I/O if the object's status information is in the metadata cache, thus a good size metadata cache can be as useful—often more useful than an extremely large vnode cache. <strong>Total osi_getvnode Calls</strong> is the number of times zFS called the osi_getvnode interface of z/OS UNIX to get a z/OS UNIX vnode to correspond to a new zFS vnode. Its <strong>high resp</strong> is the number of calls that took longer than a second to complete. <strong>Avg. Call Time</strong> is the average number of milliseconds each call took to complete. <strong>Total SAF Calls</strong> is the number of calls zFS made to the security product via the SAF interface. <strong>high resp</strong> is the number of these security calls that took longer than a second to complete. <strong>Avg. Call Time</strong> is the average number of milliseconds each call took to complete.</td>
</tr>
<tr>
<td>Field name</td>
<td>Contents</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>zFS Fast Lookup Statistics:</td>
<td>Shows the basic performance characteristics of the zFS fast lookup cache. The fast lookup cache is used on the owning system for a zFS sysplex-aware file system to improve the performance of the lookup operation. There are no externals for this cache (other than this display). The statistics show the total number of buffers (each are 8K in size), the total number of lookups, the cache hits for lookups and the hit ratio. The higher the hit ratio, the better the performance.</td>
</tr>
<tr>
<td>Metadata Caching Statistics:</td>
<td>Shows the basic performance characteristics of the metadata cache. The metadata cache contains a cache of all disk blocks that contain metadata and any file data for files less than 7 K in size. For files smaller than 7 K, zFS places multiple files in one disk block (for zFS a disk block is 8 K bytes). Only the lower metadata management layers have the block fragmentation information, so the user file I/O for small files is performed directly through this cache rather than the user file cache. The statistics show the total number of buffers (each are 8 K in size), the total bytes, the request rates, hit ratio of the cache, Updates (the number of times an update was made to a metadata block), and Partial writes (the number of times that only half of an 8-K metadata block needed to be written). The higher the hit ratio the better the performance. Metadata is accessed frequently in zFS and all metadata is contained only (for the most part) in the metadata cache therefore, a hit ratio of 80% or more is typically sufficient.</td>
</tr>
<tr>
<td>Metadata Backing Cache Statistics:</td>
<td>Describes the performance of the extension to the metadata cache. The size of this extension is controlled by the metaback_cache_size configuration option. The backing cache is stored in a dataspace and is used only to avoid metadata reads from disk. All metadata updates and write I/O are performed from the primary metadata cache. Similar statistics to the metadata cache are shown for this cache. Every hit in this cache avoids one disk read, but the metadata backing cache is not needed except for workloads with many small user files or that are constrained in the zFS primary address space (possibly because of a large demand of zFS vnodes made by z/OS UNIX and its applications). Thus, if the zFS address space has primary space available, the space should be given to the primary metadata cache. In the preceding report example, the metadata backing cache is providing little performance benefit (as shown by its small hit ratio). It can only be created by specifying the metaback_cache_size configuration option of the IOEFSPRM file or the zfsadm config command.</td>
</tr>
</tbody>
</table>
### Table 5. LFS report sections (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Cache</td>
<td>zFS updates metadata on disk by writing the changes to the metadata to a log file. Each operation will create one or more transactions, write the updates to the logs associated with the transaction and then end the transaction. Each transaction has an associated state, which is described as follows:</td>
</tr>
<tr>
<td><strong>Active</strong></td>
<td>Records are still being written to the log file to describe the updates being made by this transaction; hence, the transaction was started but has not yet ended. Shown as &quot;Act&quot; in the report.</td>
</tr>
<tr>
<td><strong>Complete</strong></td>
<td>The transaction has ended, all updates were written to the log file, and the end transaction record is also written to the log for that transaction. Shown as Comp in the report.</td>
</tr>
<tr>
<td><strong>Committed</strong></td>
<td>The transaction has ended and all updates are written to the log file and all the log file pages that contain information about this transaction are on disk. At this point, the transaction is guaranteed. The update is present if the system stopped. (In the sample report, statistics for this count is not shown. As soon as a transaction is committed, the structure representing the transaction is &quot;free&quot; for reuse for another transaction.)</td>
</tr>
<tr>
<td><strong>Equivalence Classes</strong></td>
<td>zFS does not use a common technique called two-phase locking or commit. Rather, transactions that are related are grouped into equivalence classes. zFS will decide when a transaction is related to, or dependent on, another transaction. When this determination is made, the transactions are grouped into an equivalence class. Any transactions in the same equivalence class are committed together, or backed out together, in the event of a system failure. By using equivalence classes, threads running transactions run in parallel without added serialization between the two (other than locks if they hit common structures) and add their associated transactions to the same class. This increases throughput. The merge of equivalence classes occurs when two transactions that need to be made equivalent are both already in equivalence classes. In this case, both classes are merged &quot;EC Merges&quot;.</td>
</tr>
<tr>
<td><strong>Pending</strong></td>
<td>A transaction is pending when all its updates are written to the log file, but other transactions in its same equivalence class have not ended. Shown as &quot;Pend&quot;.</td>
</tr>
</tbody>
</table>

The transaction cache size is by default 2000 transactions. It can be changed by the `tran_cache_size` configuration option. In general, zFS increases the size of the cache if it determines too many I/O waits are occurring to sync log file pages to commit transactions so that their structure can be freed and thereby improve performance. Also, if you are using the `zfsadm config` command to set the `tran_cache_size`, the transaction cache will not be shrunk too small as to cause excessive log file syncs and you will see a failure if you attempt to set the cache too small. As a rule of thumb, the default should be fine for most customers. If zFS determines more are needed for performance, it will allocate more. zFS is conservative about adding more transaction structures. You might get a small performance boost by starting with a larger transaction cache size, so that zFS does not need to make checks to determine if it can increase the size or sync log file pages.
Table 5. LFS report sections (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>zFS I/O by Currently Attached Aggregate:</td>
<td>The zFS I/O driver is essentially an I/O queue manager (one I/O queue per DASD). It uses Media Manager to issue I/O to VSAM data sets. It generally sends no more than one I/O per DASD volume to disk at one time. The exception is parallel access volume (PAV) DASD. These DASD often have multiple paths and can perform multiple I/O in parallel. In this case, zFS will divide the number of access paths by two and round any fraction up. (For example, for a PAV DASD with five paths, zFS will issue, at the most, three I/Os at one time to Media Manager). zFS limits the I/O because it uses a dynamic reordering and prioritization scheme to improve performance by reordering the I/O queue on demand. Thus, high priority I/Os (I/Os that are currently being waited on, for example) are placed up front. An I/O can be made high priority at any time during its life. This reordering has been proven to provide the best performance, and for PAV DASD, performance tests have shown that not sending quite as many I/Os as available paths allows zFS to reorder I/Os and leave paths available for I/Os that become high priority. Another feature of the zFS I/O driver is that by queueing I/Os, it allows I/Os to be canceled. For example, this is done in cases where a file was written, and then immediately deleted. Finally, the zFS I/O driver merges adjacent I/Os into one larger I/O to reduce I/O scheduling resource, this is often done with log file I/Os because often times multiple log file I/Os are in the queue at one time and the log file blocks are contiguous on disk. This allows log file pages to be written aggressively (making it less likely that users lose data in a failure) and yet batched together for performance if the disk has a high load.</td>
</tr>
<tr>
<td></td>
<td>This section contains the following information:</td>
</tr>
<tr>
<td></td>
<td>• PAV IO, which shows how many I/Os are sent in parallel to Media Manager by zFS, non PAV DASD always shows the value 1.</td>
</tr>
<tr>
<td></td>
<td>• DASD VOLSER for the primary extent of each aggregate and the total number of I/Os and bytes read/written.</td>
</tr>
<tr>
<td></td>
<td>• Number of times a thread processing a request must wait on I/O and the average wait time in milliseconds is shown</td>
</tr>
<tr>
<td></td>
<td>By using this information with the KN report, you can break down zFS response time into what percentage of the response time is for I/O wait. To reduce I/O waits, you can run with larger cache sizes. Small log files (small aggregates) that are heavily updated might result in I/Os to sync metadata to reclaim log file pages resulting in additional I/O waits. Note that this number is not DASD response time. It is affected by it, but it is not the same. If a thread does not have to wait for an I/O then it has no I/O wait; if a thread has to wait for an I/O but there are other I/Os being processed, it might actually wait for more than one I/O (the time in queue plus the time for the I/O).</td>
</tr>
<tr>
<td></td>
<td>This report, along with RMF DASD reports and the zFS FILE report, can be also used to balance zFS aggregates among DASD volumes to ensure an even I/O spread.</td>
</tr>
</tbody>
</table>

zFS Large Fast Lookup Statistics
--------------------------------
Number of Large FLC Buffers allocated 60
Number of Large FLC Buffers stolen 18
Number of Large FLC Buffers assigned 326
Number of Large FLC Buffers requests 5989235
Number of Large FLC Buffers in use 0
Number of Large FLC Hash Table Slots 4096
Number of Pieces in Large FLC 32512

If you define the IOEPRMxx configuration option f1c, the LFS query report will include statistics for the new Large FLC buffers. It contains information that includes the total number of Large FLC buffers allocated, the number of times a buffer is stolen from a directory, the number of times a buffer is assigned (and

84 z/OS V2R1.0 Distributed File Service zFS Administration
populated by reading in the entire contents of the directory) to a directory, the total number of operations (requests such as lookup, create or remove) performed on a directory that has a large buffer assigned to it, the size of the hash table in the buffer and how many name pieces each buffer contains.

The number of buffers that are allocated is the number of large buffers requested on the IOEFSRNM variable flc bufnum setting.

The number of buffers stolen is the number of times a buffer is stolen from a large directory. Stealing it can occur if z/OS UNIX inactivates the directory, zFS steals the vnode extension from the directory vnode, the containing file system is unmounted, the large directory itself is deleted or if the large directory has not been accessed in the number of seconds specified by the IOEFSRNM variable flc inactive setting. If this number seems high, it might be because the directory is not being accessed frequently. If it is being accessed frequently, then try increasing the vnode cache size or increasing the IOEFSRNM variable flc inactive value.

Number of buffers assigned is the number of times a large directory that does not currently have a Large FLC buffer assigned to it and has a size greater than or equal to IOEFSRNM variable flc mindirsize is accessed. This causes a Large FLC buffer to be populated by reading the contents of the entire directory from disk. If this number seems high, it might be because the directory is not being accessed frequently. If it is being accessed frequently, then try increasing the vnode cache size or increasing the IOEFSRNM variable flc inactive value. Another possible cause is that the IOEFSRNM variable flc mindirsize value is too small. This could occur if, over time, directories have grown in size and now more are considered to be large. If this has happened, then increase the IOEFSRNM variable flc mindirsize value to a more appropriate value or increase the IOEFSRNM variable flc bufnum to allocate more Large FLC buffers. If this number seems low, you might need to decrease the IOEFSRNM variable flc inactive to allow for more frequent Large FLC buffer reuse.

The number of buffer requests is the number of lookup, create, or remove directory operations done to a directory with a Large FLC buffer. This number can be used as an indicator of how frequently the large directories are being accessed.

The number of buffers in use is the number of Large FLC buffers that are currently assigned to a large directory. If you repeatedly find that this number is much lower than, or nearly the same as, the number of allocated buffers then you should verify that your IOEFSRNM variable flc bufnum setting is correct. Directories that used to exist might have been deleted or other (possibly new) directories might have grown to IOEFSRNM variable flc mindirsize and are now considered large.

Also included is the number of slots in the hash table and the number of pieces in the Large FLC buffers. Message IOEZ00821E indicates that a directory contains too many entries to have a Large FLC buffer assigned to it. Message IOEZ00819E indicates that a directory had a Large FLC buffer assigned to it, but a new entry was added to the directory that caused it to become too big. These messages indicate that the hash table and number of pieces are not sufficient. The IOEFSRNM variable flc bufnum value needs to be increased. This could have occurred because over time, the directories grew larger than previously expected. It is very important to reevaluate the IOEFSRNM variable flc settings periodically.

LOCK
The LOCK report is mainly for IBM service to use when diagnosing performance problems relating to lock contention. The report shows a detailed breakdown of
how often zFS waits for locks and which locks cause the most contention. It also monitors how often a thread sleeps waiting for an event. The lock waits, lock wait time, sleep waits, and sleep wait time can be used with the KN report to break down zFS response time into what percentage of the time zFS is waiting on internal locks or events to occur. By extrapolation, you can guess that the remaining time is CPU time and processor wait time. Figure 37 on page 87 shows an example of a report.
LOCK:

Locking Statistics

Untimed sleeps: 22 Timed Sleeps: 0 Wakeups: 21

Total waits for locks: 3698
Average lock wait time: 8.261 (msecs)

Total monitored sleeps: 22
Average monitored sleep time: 0.792 (msecs)
Total starved waiters: 0
Total task priority boosts: 0

Top 15 Most Highly Contended Locks

<table>
<thead>
<tr>
<th>Thread</th>
<th>Async Spin</th>
<th>Wait Disp.</th>
<th>Resol.</th>
<th>Pct.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>877</td>
<td>0</td>
<td>899</td>
<td></td>
<td>35.763%</td>
<td>Log system map lock</td>
</tr>
<tr>
<td>1464</td>
<td>0</td>
<td>40</td>
<td>28</td>
<td>30.285%</td>
<td>Anode bitmap allocation handle</td>
</tr>
<tr>
<td>481</td>
<td>0</td>
<td>28</td>
<td>42</td>
<td>10.249%</td>
<td>Anode fileset quota lock</td>
</tr>
<tr>
<td>291</td>
<td>0</td>
<td>42</td>
<td>62</td>
<td>6.705%</td>
<td>Transaction lock</td>
</tr>
<tr>
<td>205</td>
<td>0</td>
<td>62</td>
<td>4</td>
<td>5.376%</td>
<td>Metadata-cache buffer lock</td>
</tr>
<tr>
<td>210</td>
<td>0</td>
<td>4</td>
<td></td>
<td>4.309%</td>
<td>Anode fileset handle lock</td>
</tr>
<tr>
<td>84</td>
<td>68</td>
<td>7</td>
<td>3.201%</td>
<td>1.107%</td>
<td>User file cache main segment lock</td>
</tr>
<tr>
<td>0</td>
<td>55</td>
<td>0</td>
<td></td>
<td>0.765%</td>
<td>Volser I/O queue lock</td>
</tr>
<tr>
<td>38</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>0.724%</td>
<td>Transaction-cache main lock</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>11</td>
<td>0</td>
<td>0.443%</td>
<td>Transaction-cache equivalence c</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0.422%</td>
<td>Async I/O event lock</td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>0</td>
<td>5</td>
<td>0.281%</td>
<td>Cache Services association main</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0.120%</td>
<td>Cache Services hashtable resize</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td></td>
<td>0.100%</td>
<td>Transaction-cache complete list</td>
</tr>
</tbody>
</table>

Total lock contention of all kinds: 4966

Top 5 Most Common Thread Sleeps

<table>
<thead>
<tr>
<th>Thread Wait</th>
<th>Pct.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>100.000%</td>
<td>Transaction allocation wait</td>
</tr>
<tr>
<td>0</td>
<td>0.000%</td>
<td>OSI cache item cleanup wait</td>
</tr>
<tr>
<td>0</td>
<td>0.000%</td>
<td>Directory Cache Buffer Wait</td>
</tr>
<tr>
<td>0</td>
<td>0.000%</td>
<td>User file cache Page Wait</td>
</tr>
<tr>
<td>0</td>
<td>0.000%</td>
<td>User file cache File Wait</td>
</tr>
</tbody>
</table>

Example:

From the KN report we get the following:
Total zFS requests: 91905
Avg. Resp. Time: 1.108

From the LFS report we get:
Total I/O waits: 556
Avg. I/O wait time: 62.215

Avg. I/O wait time per request = 556/91905 * 62.215 = 0.376
(this is 34% of the response time (0.376/1.108=.34)).

From the locking report we get:
Total Waits for Locks: 3698
Avg. Lock wait time: 8.261

Avg. Lock wait time per request = 3698/91905 * 8.261 = 0.332
(this is 30% of the response time (0.332/1.108=.30)).

Figure 37. Example of a LOCK report
LOG

The Figure 38 shows performance statistics for the Log File Cache. The Log File Cache is a write-only cache that is stored in a dataspace and is shared among all attached R/W aggregates that are zFS-owned on a system. zFS will ensure that is at least one Log File Buffer for each aggregate it represents, so modifying IOEFSPRM configuration option log_cache_size to change the size of the cache should not be necessary.

In the report, **Buffers** is the number of 8 K buffers in the cache, **K bytes** is the size of the cache in K units, **Requests** is the number of updates made to a log cache buffer, **Hits** is the number of log cache lookups or obtains that were found to be in the cache already, **Ratio** is the percentage of **Ratios** that are **Hits**, **Written** is the number of log buffers (log pages) written to disk, **log full waits** is the number of times a task had to wait for a new log buffer because all log buffers were being written to disk and **NBS IO waits** is the number of times writing a log page had to wait for user I/O to be completed first.

If the number of log full waits for new buffers is high, you might try adding log buffers to the cache by increasing the log_cache_size. This action essentially stops all operations in all file systems, so care should be taken to not try to grow the cache during peak times or by too large an amount of buffers.

---

<table>
<thead>
<tr>
<th>Buffers (K bytes)</th>
<th>Requests</th>
<th>Hits</th>
<th>Ratio</th>
<th>Written</th>
</tr>
</thead>
<tbody>
<tr>
<td>2050</td>
<td>16400</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*Figure 38. Sample LOG report*

STKM

The STKM report, as shown in Figure 39 on page 89 lists the server token manager statistics. LOCALUSR is the local system (the server). ZEROLINK is a “special client” used to handle zero link count files and vnode inactivations. Table 6 on page 89 describes the contents of the report.
### Table 6. STKM report fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum tokens:</td>
<td>Lists the token limit at the server which is defined by the IOEFSPRM configuration option <code>token_cache_size</code>. The server runs garbage collection to ensure that token maximum is not exceeded. In some cases, the system workload might cause the token maximum to be exceeded, such as when there are many open files.</td>
</tr>
<tr>
<td>Allocated tokens:</td>
<td>Number of tokens allocated in server memory. Tokens are allocated as needed, up to <code>maximum</code> tokens.</td>
</tr>
<tr>
<td>File structures:</td>
<td>Number of file structures.</td>
</tr>
<tr>
<td>Tokens In Use:</td>
<td>Number of tokens currently held by all clients and the local system. If this number approaches maximum tokens, then consider increasing the <code>token_cache_size</code> setting.</td>
</tr>
<tr>
<td>Token obtains:</td>
<td>Total number of token obtains by all clients and local system.</td>
</tr>
<tr>
<td>Token revokes:</td>
<td>Total number of token revokes by all clients and local system.</td>
</tr>
<tr>
<td>Token returns:</td>
<td>Total number of token returns by all clients and local system.</td>
</tr>
<tr>
<td>Async grants:</td>
<td>Number of asynchronously granted tokens to all clients and local system. Asynchronous grant is used during file deletion processing when the file is still opened by some process in the sysplex, and in support of NFS V4 share modes.</td>
</tr>
<tr>
<td>Garbage collects:</td>
<td>Number of garbage collections of tokens. Garbage collection is used to keep the total number of client/local system tokens below the maximum whenever possible. If this number gets high, consider increasing the <code>token_cache_size</code> setting.</td>
</tr>
<tr>
<td>Thrashing files:</td>
<td>Number of file systems that are thrashing.</td>
</tr>
<tr>
<td>Thrashing resolutions:</td>
<td>Number of thrashing situations resolved.</td>
</tr>
</tbody>
</table>

### STOR

The STOR report provides a breakdown of zFS storage usage. This report can be used to determine how much storage zFS uses, based on a configuration change (such as increasing or decreasing a zFS cache through the `zfsadm config` command). Figure 40 on page 91 shows a sample report and Table 7 on page 92 explains the contents of each field. (Not shown here is the output of `QUERY,STOR,DETAILS`, which breaks down each component and shows how much storage is used for each data structure class; this report is intended primarily for IBM service.)
It is good practice to periodically check zFS storage usage by issuing the operator command `modify zfs,query,storage`. If you compare the third line of data (USS/External Storage Access Limit) to the fourth line (Total Bytes Allocated (Stack+Heap+OS)), you can determine how close zFS is to using its maximum storage. The Total Bytes Allocated should be less than the USS/External Storage Access Limit. For example, in Figure 40 on page 91, the Total Bytes Allocated (254M) is much less than the USS/External Storage Access Limit (1800M).

If the Total Bytes Allocated becomes greater than or equal to the USS/External Storage Access Limit, zFS will issue message IOEZ00662I. If the Total Bytes Allocated approaches the value of the USS/External Storage Access Limit, you can attempt to dynamically decrease the caches using the `zfsadm config` command. (You should also make the corresponding changes in your IOEFSPRM file for the next zFS restart.) Alternatively, you can stop and restart zFS after making cache size changes to your IOEFSPRM file.

If zFS has failed to initialize and is not active, you should decrease some of your zFS IOEFSPRM settings, especially if they are significantly larger than the default values, and restart zFS. The settings to review include:

- `meta_cache_size`
- `recovery_max_storage`
- `token_cache_size`
- `tran_cache_size`
- `vnode_cache_size`

If zFS is active but message IOEZ00662I has been issued, you can issue the `zfsadm config` command to attempt to decrease the cache sizes dynamically. You should also make the corresponding changes in your IOEFSPRM file for the next zFS restart. Alternatively, you can stop and restart zFS after making cache size changes to your IOEFSPRM file.

The IOEFSCM storage type fields are displayed only when the efskn debug class is active.

To represent 64 bit numbers in the part of the storage report that shows zFS storage usage above the 2 G bar, we are adopting a representation of the form `<x,,y>`. Think of a number that is above the 2 G bar, for example 0x0004000400000010. This can be thought of in two separate words, 0x00040004 and 0x00000010. Let x be the first half, 0x00040004 and y be the second half, 0x00000010. Now, think of the two number x and y in decimal. This would give <262148,,16>. Position
### zFS Primary Address Space Storage Usage

- **Total Storage Available to zFS**: 1950351360 (1904640K) (1860M)
- **Non-critical Storage Limit**: 1929379840 (1884160K) (1840M)
- **USS/External Storage Access Limit**: 1887436800 (1843200K) (1800M)
- **Total Bytes Allocated (Stack+Heap+OS)**: 266702848 (260452K) (254M)
- **Heap Bytes Allocated**: 234760807 (229258K) (223M)
- **Heap Pieces Allocated**: 218501
- **Heap Allocation Requests**: 226154
- **Heap Free Requests**: 7653

#### Heap Usage By Component

<table>
<thead>
<tr>
<th>Bytes Allocated</th>
<th>No. of Pieces</th>
<th>No. of Allocs</th>
<th>No. of Frees</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>67348</td>
<td>18</td>
<td>18</td>
<td>0</td>
<td>Interface</td>
</tr>
<tr>
<td>46824</td>
<td>29</td>
<td>118</td>
<td>89</td>
<td>Media Manager I/O driver</td>
</tr>
<tr>
<td>71304984</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>Trace Facility</td>
</tr>
<tr>
<td>401876</td>
<td>3</td>
<td>2428</td>
<td>2425</td>
<td>Message Service</td>
</tr>
<tr>
<td>318621</td>
<td>1212</td>
<td>1218</td>
<td>6</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>29400</td>
<td>88</td>
<td>118</td>
<td>30</td>
<td>Aggregate Management</td>
</tr>
<tr>
<td>113512</td>
<td>112</td>
<td>150</td>
<td>38</td>
<td>Filesystem Management</td>
</tr>
<tr>
<td>33996</td>
<td>23</td>
<td>74</td>
<td>51</td>
<td>Administration Command Handling</td>
</tr>
<tr>
<td>15472609</td>
<td>98355</td>
<td>98363</td>
<td>8</td>
<td>Vnode Management</td>
</tr>
<tr>
<td>15984596</td>
<td>33498</td>
<td>34162</td>
<td>664</td>
<td>Anode Management</td>
</tr>
<tr>
<td>34364189</td>
<td>96</td>
<td>96</td>
<td>0</td>
<td>Directory Management</td>
</tr>
<tr>
<td>476752</td>
<td>6202</td>
<td>6227</td>
<td>25</td>
<td>Log File Management</td>
</tr>
<tr>
<td>34828608</td>
<td>12580</td>
<td>12605</td>
<td>25</td>
<td>Metadata Cache</td>
</tr>
<tr>
<td>420312</td>
<td>4014</td>
<td>4014</td>
<td>0</td>
<td>Transaction Management</td>
</tr>
<tr>
<td>176802</td>
<td>467</td>
<td>467</td>
<td>0</td>
<td>Asynchronous I/O Component</td>
</tr>
<tr>
<td>63724</td>
<td>196</td>
<td>200</td>
<td>4</td>
<td>Lock Facility</td>
</tr>
<tr>
<td>194364</td>
<td>468</td>
<td>468</td>
<td>0</td>
<td>Threading Services</td>
</tr>
<tr>
<td>62152</td>
<td>1187</td>
<td>1199</td>
<td>12</td>
<td>Cache Services</td>
</tr>
<tr>
<td>48382</td>
<td>10</td>
<td>14</td>
<td>4</td>
<td>Configuration parameters processing</td>
</tr>
<tr>
<td>11092824</td>
<td>35506</td>
<td>35506</td>
<td>0</td>
<td>User File Cache</td>
</tr>
<tr>
<td>53432</td>
<td>90</td>
<td>2213</td>
<td>2123</td>
<td>Storage Management</td>
</tr>
<tr>
<td>42347864</td>
<td>3634</td>
<td>5793</td>
<td>2159</td>
<td>XCF Services</td>
</tr>
<tr>
<td>28584</td>
<td>16</td>
<td>26</td>
<td>10</td>
<td>Cross system attach validation</td>
</tr>
<tr>
<td>24628080</td>
<td>29529</td>
<td>209552</td>
<td>23</td>
<td>Server Token Manager (STKM)</td>
</tr>
<tr>
<td>14932</td>
<td>46</td>
<td>46</td>
<td>0</td>
<td>Server Token Cache (STKC)</td>
</tr>
<tr>
<td>4697362</td>
<td>115</td>
<td>116</td>
<td>1</td>
<td>Client Token Cache (CTKC)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Server Vnode Interface (SVI)</td>
</tr>
<tr>
<td>408</td>
<td>2</td>
<td>12</td>
<td>10</td>
<td>Name Space (NS)</td>
</tr>
<tr>
<td>1048572</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>Directory storage</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Salvage storage</td>
</tr>
</tbody>
</table>

### Figure 40. Sample STOR report (1 of 2)
Table 7. STOR report fields

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Storage Available</td>
<td>Total virtual storage in the zFS address space that is available for usage (such as caches,</td>
</tr>
<tr>
<td>to zFS</td>
<td>control blocks and stacks).</td>
</tr>
</tbody>
</table>

Figure 41. Sample STOR report (2 of 2)
<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-critical Storage</td>
<td>The value that, when exceeded, will cause zFS to issue message IOEZ00663I ZFS is critically low on storage.</td>
</tr>
<tr>
<td>Limit</td>
<td></td>
</tr>
<tr>
<td>USS/External Storage</td>
<td>The value that, when exceeded, will cause zFS to issue message IOEZ00662I ZFS is low on storage.</td>
</tr>
<tr>
<td>Access Limit</td>
<td></td>
</tr>
<tr>
<td>Total Bytes Allocated</td>
<td>The current usage of virtual storage in the zFS address space (requested by zFS and other components running in the zFS address space).</td>
</tr>
<tr>
<td>Heap Bytes Allocated</td>
<td>The current amount of storage obtained by zFS.</td>
</tr>
<tr>
<td>Heap Pieces Allocated</td>
<td>The current number of storage areas that have been requested by zFS.</td>
</tr>
<tr>
<td>Heap Allocation</td>
<td>Number of requests zFS has made to obtain storage.</td>
</tr>
<tr>
<td>Requests</td>
<td></td>
</tr>
<tr>
<td>Heap Free Requests</td>
<td>Number of requests zFS has made to free storage.</td>
</tr>
<tr>
<td>Heap Usage By Component</td>
<td>The details of the zFS heap usage by the zFS component.</td>
</tr>
</tbody>
</table>

**SVI**

The server vnode interface component handles this call. The example report that is shown in Figure 42 on page 94 displays the total number of calls the server received from the specific client and the average server response time in milliseconds, including the XCF transmit and CPU time of the reply. XCF Req is the count of XCF messages that had to be sent to other systems (most likely for token revokes) to process the client request. Qwait counts the number of times a wait was done for an available zFS thread to process the client request.

**Note:** The output is displayed only when the zFS svi component on this system has received a message from another system.
The User File (VM) Caching System Statistics report shows the performance of the zFS user file cache. The Sysplex Client Caching System Statistics Report shows the performance of the zFS client file cache. The size of both of these caches is controlled by the IOEFSPRM user_cache_size or client_cache_size configuration options or the zfsadm config command.

The zFS user file cache and the client file cache data are stored in a collection of dataspaces. zFS prefers to use multiple dataspaces rather than one large dataspace when it can to reduce lock contention (as shown in this example). zFS has a structure for each file that is cached. The client cache breaks the cached file into 32 K segments and the user cache breaks the cached file into 64 K segments. In both cases, each segment is broken into 4 K pages. A segment is assigned to a dataspace, hence the pages for any given segment belong only to one dataspace. A file's pages can be scattered throughout multiple segments.

At any given time, a file need not (and for large files often might not) have all of its segments in the cache. Furthermore, any segment need not (and often might not) have all of its pages in the cache. Reuse of pages and segments is done in an least-recently used (LRU) fashion.

### SVI Calls from System NP1

<table>
<thead>
<tr>
<th>SVI Call</th>
<th>Count</th>
<th>Qwait</th>
<th>XCF Req.</th>
<th>Avg. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetToken</td>
<td>663624</td>
<td>2</td>
<td>180593</td>
<td>4.246</td>
</tr>
<tr>
<td>GetMultTokens</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>ReturnTokens</td>
<td>814</td>
<td>0</td>
<td>0</td>
<td>8.139</td>
</tr>
<tr>
<td>ReturnFileTokens</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FetchData</td>
<td>132962</td>
<td>0</td>
<td>13222</td>
<td>1.016</td>
</tr>
<tr>
<td>StoreData</td>
<td>140171</td>
<td>9</td>
<td>0</td>
<td>0.229</td>
</tr>
<tr>
<td>Setattr</td>
<td>228600</td>
<td>0</td>
<td>0</td>
<td>0.027</td>
</tr>
<tr>
<td>FetchDir</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.188</td>
</tr>
<tr>
<td>Lookups</td>
<td>93113</td>
<td>1</td>
<td>1934</td>
<td>2.875</td>
</tr>
<tr>
<td>GetTokensDirSearch</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>Create</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5.056</td>
</tr>
<tr>
<td>Remove</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>9.040</td>
</tr>
<tr>
<td>Rename</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>Link</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>ReadLink</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>SetACL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>Stats</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0.448</td>
</tr>
<tr>
<td>TSR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FilesysSyncTable</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FileSyncMeta</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0.097</td>
</tr>
<tr>
<td>BitmapReserve</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>BitmapUnreserve</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>BitmapReclaim</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FileUpdateIB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FileCreateIB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FwdReaddir</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>LookupInvalid</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FileDebug</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>FetchPage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>ServerIO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>BulkFetchStatus</td>
<td>5563</td>
<td>0</td>
<td>0</td>
<td>4.404</td>
</tr>
<tr>
<td>Convert</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>ConvertFID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
</tbody>
</table>

*TOTALS* 2520851 12 195751 1.557

*Figure 42. Sample SVI report*
The cache provides asynchronous read-ahead and write-behind of large files when access is considered sequential. Read-ahead and write-behind for a file is performed by reading/writing segments (up to 64 K).

Following is a sample VM report.

IOEZ00438I Starting Query Command VM.

User File (VM) Caching System Statistics
----------------------------------------

External Requests:
---------------------
Reads 5107802  Fsyncs 1990  Schedules 366517
Writes 5503223  Setattrs 7091  Unmaps 349352
Asy Reads 3809619  Getattrs 1219759  Flushes 0

File System Reads:
-------------------
Reads Faulted 164132  (Fault Ratio 3.213%)
Writes Faulted 167756  (Fault Ratio 3.048%)
Read Waits 0  (Wait Ratio 0.000%)
Total Reads 357542

File System Writes:
-------------------
Scheduled Writes 119608  Sync Waits 22706
Error Writes 0  Error Waits 0
Scheduled deletes 0
Page Reclaim Writes 37315  Reclaim Waits 19392
Write Waits 5454  (Wait Ratio 0.099%)

Page Management (Segment Size = 64K) (Page Size = 4K)

Total Pages 12800  Free 12800
Segments 2373
Steal Invocations 24146  Waits for Reclaim 62038
Number of dataspaces used: 1  Pages per dataspace: 12800

Dataspace  Allocated  Free
Name  Segments  Pages
--------  ----------  ----------
ZFSUCD00 0 12800

Sysplex Client Caching System Statistics
----------------------------------------

External Requests:
---------------------
Reads 0  Fsyncs 0  Schedules 1
Writes 46737  Setattrs 0  Unmaps 0
Asy Reads 0  Getattrs 0  Flushes 0

File System Reads:
-------------------
Reads Faulted 0  (Fault Ratio 0.000%)
Writes Faulted 0  (Fault Ratio 0.000%)
Read Waits 0  (Wait Ratio 0.000%)
Total Reads 0

File System Writes:
-------------------
Scheduled Writes 1462  Sync Waits 0
Error Writes 106  Error Waits 0
Scheduled deletes 0
Page Reclaim Writes 0  Reclaim Waits 0
Write Waits 0  (Wait Ratio 0.000%)

Page Management (Segment Size = 32K) (Page Size = 4K)
Table 8. User File (VM) Caching System Statistics report fields

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Requests:</td>
<td>Describes the requests made to the user file cache to perform operations as requested by applications.</td>
</tr>
<tr>
<td>Reads, Writes</td>
<td>How often the cache was called to read or write files.</td>
</tr>
<tr>
<td>Asy Reads</td>
<td>How often read-ahead is performed.</td>
</tr>
<tr>
<td>Fsync</td>
<td>How often applications requested that zFS sync a file’s data to disk.</td>
</tr>
<tr>
<td>Unmaps</td>
<td>The count of file deletions.</td>
</tr>
<tr>
<td>File System Reads:</td>
<td>Shows how often the cache reads data from disk for a file. Cache misses and read I/Os degrade application response time and the goal is for these numbers to be as low as possible. Increasing the cache size is the typical method for lowering these numbers.</td>
</tr>
<tr>
<td>Reads Faulted</td>
<td>Count of read requests that needed to perform at least one I/O to read the requested portion of the file from disk.</td>
</tr>
<tr>
<td>Writes Faulted</td>
<td>Count of how often a write to a file needed to perform a read from disk. If a write only updates a portion of a page of a file on disk and that page is not in memory, then the page must be read in (the zFS I/O driver can only perform I/O in whole pages) before the new data is written to the in-memory page.</td>
</tr>
<tr>
<td>Read Waits</td>
<td>How often a read had to wait for a pending I/O. For example, how often a read of a file found that the range of the file is pending read (probably because of asynchronous read ahead).</td>
</tr>
<tr>
<td>Total Reads</td>
<td>Total number of file system reads made for any reason.</td>
</tr>
<tr>
<td>Field name</td>
<td>Contents</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>File System Writes:</td>
<td>Shows how often the cache wrote the data to disk. In general, it is desirable to minimize the Page Reclaim Writes and Reclaim Waits. If these occur often, relative to the external zFS request rate (shown in the KN report), then the cache might be too small.</td>
</tr>
<tr>
<td>Scheduled Writes</td>
<td>Count of how often the cache wrote out dirty segments for a file. Segments are written as soon as every page becomes dirty (segments are said to be dirty if they contain live blocks). When a file is closed all of its dirty segments are scheduled asynchronously and segments are also written asynchronously during file system syncs through the zFS sync daemon (which by default runs every 30 seconds).</td>
</tr>
<tr>
<td>Sync Waits</td>
<td>Count of how often a fsync request that is needed to wait on pending I/O for dirty segments.</td>
</tr>
<tr>
<td>Error Writes and Error Waits</td>
<td>Count of the error handling paths and should almost always be 0 unless a disk hardware error occurs. Whenever an unexpected error occurs for a file, all of its dirty segments are written and synced to disk. (A file system that is running out of space is not an error condition that causes the cache to sync a file, the cache reserves storage for files as they are written which ensures no unexpected out of space conditions arise).</td>
</tr>
<tr>
<td>Scheduled Deletes</td>
<td>Count of times a pending I/O was canceled because a file was being deleted. In this case, the data is not appropriate to be on disk (because the file is 0 link count). Therefore, canceling the I/O is done to avoid an I/O wait. This is a performance optimization for file remove.</td>
</tr>
<tr>
<td>Page Reclaim Writes</td>
<td>Count of times that a segment had to be written to DASD to reclaim space in the cache</td>
</tr>
<tr>
<td>Page Reclaim Waits</td>
<td>Count of times that the reclaim function waited on pending I/O to reclaim segment pages.</td>
</tr>
<tr>
<td>Write Waits</td>
<td>Count of times a write occurred to a page that was already pending I/O. In this case, the I/O must be waited upon before the page is updated with the new data.</td>
</tr>
</tbody>
</table>
Table 8. User File (VM) Caching System Statistics report fields  (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Management:</td>
<td>Shows how storage in the user file cache and client file cache is used. It is generally desirable to minimize the number of steal invocations (reclaims). This can typically be done by increasing the size of the cache. Performance is increased as more dataspaces are used.</td>
</tr>
<tr>
<td><strong>Total pages</strong></td>
<td>The number of 4 K pages in the cache. That is, (\frac{\text{user_cache_size}}{4\text{K}}) or (\frac{\text{client_cache_size}}{4\text{K}}).</td>
</tr>
<tr>
<td><strong>Free</strong></td>
<td>The number of available 4 K pages in the cache.</td>
</tr>
<tr>
<td><strong>Segments</strong></td>
<td>The number of 64 K sections that was referenced in a file. The number of segments starts out as half of vnode_cache_size and is allocated as needed, similar to vnodes.</td>
</tr>
<tr>
<td><strong>Steal Invocations</strong></td>
<td>The number of times 4 K pages were reclaimed from the cache.</td>
</tr>
<tr>
<td><strong>Waits for Reclaim</strong></td>
<td>The number of times 4 K pages were reclaimed from the cache.</td>
</tr>
<tr>
<td><strong>Number of dataspaces</strong></td>
<td>The number of dataspaces used to hold the 4 K pages in the cache. The pages are spread evenly across the dataspaces to allow for better performance of the cache. The number of dataspaces used is approximately one per 16384 4 K pages, up to a maximum of 32.</td>
</tr>
<tr>
<td><strong>Pages per dataspace</strong></td>
<td>The number of 4 K pages that is assigned to each dataspace.</td>
</tr>
</tbody>
</table>

Debugging aids for zFS

If a problem occurs in zFS that requires the attention of IBM support, it is important to obtain the appropriate problem determination information to help resolve the problem quickly. This section covers topics to help you gather this information.

Overview of trace options for zFS

One of the most important aspects of zFS problem determination is its tracing capability. zFS has an internal (wrap around) trace table that is always tracing certain events. The size of this trace table is controlled by the IOEFSPRM trace_table_size option.

Steps for tracing on zFS

If you are re-creating a problem and need to collect a zFS trace, use the following steps:

1. Allocate the trace output data set as a PDSE, RECFM=VB, LRECL=133 with a primary allocation of at least 50 cylinders and a secondary allocation of 30 cylinders.

2. Define the zFS trace output data set to zFS by either using the IOEFSPRM trace_dsn option, or dynamically by using the `zfsadm config -trace_dsn` command.

If you use the IOEFSPRM option, zFS must be stopped and then restarted to pick up the change, unless you also dynamically activate the trace output data set with the `zfsadm config -trace_dsn` command.
3. When you are ready to re-create the problem, reset the zFS trace table using the MODIFY ZFS,TRACE,RESET command.
4. Re-create the problem.
5. Enter the MODIFY ZFS,TRACE,PRINT command. This formats and prints the trace table to the PDSE defined on the trace_dsn option.
6. Capture the ZFSKNTnn member from the trace output data set, (for example, copy it to a sequential data set) so that it can be sent to IBM service.

A separate trace output data set is required for each member of a sysplex.
1. Ensure that you set up the trace data sets so that each system in the sysplex can write to its own trace output data set concurrently. This requires separate IOEFSPRM files or the use of system symbols in the trace_dsn name or the use of an IOEPRMxx parmib member. For more information, see Chapter 5, “Using zFS in a shared file system environment,” on page 51.
2. Allocate the data set as a PDSE, RECFM=VB, LRECL=133 with a primary allocation of at least 50 cylinders and a secondary allocation of 30 cylinders. Each trace output is created as a new member with a name of ZFSKNTnn, where nn starts at 01 and increments for each trace output until zFS is restarted. After restart, when the next trace output is sent to the trace output data set, ZFSKNT01 is overlaid. You should not be accessing the trace output data set while a trace is being sent to the trace output data set. The space that is used by a particular trace depends on how large the trace_table_size is and how recently the trace was reset. For example, a 32-MB trace_table_size can generate a trace output member of 100 cylinders of 3390. It is important that the trace output data set be large enough to hold the trace output. If it runs out of room while sending the trace to the trace output data set, the complete trace will not be captured.

Note: You can have a trace_table_size up to 2048 MB, but to print the trace to a PDSE you must limit its size to 750 MB.

IBM service might require you to trace more events. Additional trace information can be obtained using the following methods:

- Add events to trace by specifying the ioedebug statements in a data set that is read when zFS is started (or restarted). The data set name is specified in the IOEFSPRM debug_settings_dsn option. It is a PDS member with an LRECL of at least 80. IBM specifies the exact statements needed in the data set.
- Dynamically add the events to trace by entering the MODIFY ZFS,IOEDEBUG command. IBM specifies the exact statements needed.
- If you were not able to capture the trace, but you have a zFS dump, IBM service can obtain the trace from the dump. To obtain a dump, you can issue a MODIFY ZFS command. See “Understanding zFS dumps” on page 101 for additional information.

The zFS trace table is above the 2-GB bar to avoid consuming space in the zFS address space, which is below the bar.

**Understanding the salvager utility**

The salvager (ioeagslv or ioefsutl salvage) utility is a zFS-supplied program that runs as a batch job. It examines a zFS aggregate to determine if there are any inconsistencies in the structure of the aggregate. In many cases, it can also fix a corrupted aggregate. Before running the salvager utility against an aggregate, the aggregate must be unmounted (detached). When a zFS aggregate is not cleanly
unmounted (for example, system is re-IPLed without a shutdown, system goes
down, zFS abends and goes down, zFS is canceled, and so on), the next time the
aggregate is mounted, zFS will play the aggregate log to bring the aggregate back
to a consistent state. Message IOEZ00397I (among others) is issued to indicate zFS
is playing the log. Usually, running the log is successful and does not require any
other action. However, even though the aggregate is consistent, you can still have
some data loss if information was being written shortly before or at the time the
failure occurred.

There are times, listed in the following list, when it might be appropriate to run
the salvager utility against a zFS aggregate. Depending on how the file system is
used at your installation, you might want to run the salvager to ensure that there
is no corruption or to attempt to correct a corruption. For example, if the file
system has not yet been mounted or you can take it offline without impacting
many users or applications, you might want to run the salvager soon after the
problem occurs. Conversely, if the file system is used extensively, you might decide
decide not to run the salvager or wait for a more convenient time to do so.

• An internal error has occurred during zFS processing for the aggregate.
  In this situation, zFS issues abend 2C3 and message IOEZ00422E. zFS detected a
  problem and disabled the aggregate so that no reads or writes can occur for this
  aggregate until it is remounted. This action attempts to avoid writing incorrect
  data that might corrupt the aggregate. If you want to run the salvage utility, you
  must first unmount the aggregate.

• An I/O error has occurred while accessing the aggregate. zFS detected a
  physical I/O error on the device.
  In this case, zFS issues messages IOEZ00001E or IOEZ00550E and the message
  IOEZ00422E. zFS detected the I/O error and disabled the aggregate. This is most
  likely a hardware problem. Follow your local procedures for analyzing I/O
  problems to determine if you want to run the salvage utility. If you run the
  utility, you must first unmount the aggregate.

• A zFS problem occurs during a mount of a zFS aggregate.
  zFS detected a problem while mounting a zFS aggregate. The mount might
  receive a return code of EMVSERR (decimal 157). zFS might issue a
  non-terminating abend during the mount. In this case, you might choose to run
  the salvager because the aggregate was not yet mounted.

If an aggregate cannot be repaired successfully, the salvager marks it as damaged.
If it is then mounted, an IOEZ00783E message is issued indicating that a damaged
aggregate was mounted.

If you decide to run the salvager utility, specify the -verifyonly option to examine
the aggregate structures. If there are no error messages, the aggregate is not
corrupted. If you run the salvager utility with no options, it attempts to fix any
corruptions that it finds.

In the following situations, the salvager utility might not always be able to fix a
corrupted aggregate:

• If a fundamental aggregate structure is corrupted, the salvager will not be able
to recover the aggregate.

• If the aggregate is large or has many objects, the salvager might not be able to
  complete successfully. Even when the salvager is successful, an aggregate with
  many objects will take a long time to examine and attempt to repair. It might
take less time to restore a backup copy of the aggregate than to salvage it.
The salvager is designed to make all repairs in one pass, but due to the nature of the program's inputs (a corrupted, possibly vastly corrupted file system) IBM recommends a second running of the salvage program to verify that the aggregate is truly repaired. If verifying the aggregate shows that it is not repaired, then you should try running the salvager again to repair the aggregate. If this does not repair the aggregate, you can create a copy of the aggregate and run the salvager more times to try and repair it. If the salvager cannot repair the aggregate after several repair attempts, the copy of the aggregate and salvager job logs will allow IBM service to determine why.

It is important to maintain backups of zFS aggregates to restore in case of a corrupted aggregate. It is also very important to maintain a regular backup regimen (for example, daily, weekly, monthly) so that if a recent backup is corrupted, you can use an older backup. However, if a quiesce is not done before backup, corruption of the file system can result. See Chapter 6, “Performing a backup of zFS,” on page 63 for recommendations for backing up zFS aggregates.

**Understanding zFS dumps**

Another important source of information is a zFS dump. Any time a zFS failure occurs, you should check the system log to see if zFS has performed a dump. In a sysplex, zFS typically requests a dump on the other sysplex members; check to see if other members have zFS dumps. Typically, these will have the following message:

10E200337E zFS kernel: non-terminating exception 2C3 occurred, reason EA2F0385

The abend reason of EAxx0385 indicates that the dump was requested by zFS from another sysplex member. If zFS does not automatically request a dump from the other sysplex members, you should enter the MODIFY ZFS,DUMP command on these other systems.

zFS also sends the trace to the trace output data set when a zFS dump occurs. Note that when a zFS abend occurs, other application failures might occur. For problem determination, these failures are not as important as the original zFS failure and dump.

Typically, zFS does not stop as a result of a zFS failure. An aggregate might become disabled (see “Diagnosing disabled aggregates” on page 110). If zFS does stop, zFS attempts to perform an internal restart after the terminating exception occurs. If the internal restart is unsuccessful, zFS attempts a stop and restart sequence. If the restart is successful, you might need to remount any zFS file systems. The SETOMVS command can be used to remount file systems that were mounted from a BPXPRMxx parmlib member statement.

If a failure of a zFS operation occurs (other than a user error), but zFS does not dump, you should get a trace of the failure, if possible. Perform the steps outlined in “Steps for tracing on zFS” on page 98.

You can also obtain a dump of the zFS address space by entering the MODIFY ZFS,DUMP command. The dump should contain the zFS trace table. You must ensure the dump is complete. Partial dumps are of little use.

Alternatively, you can enter the MODIFY ZFS,ABORT command to cause zFS to send the trace to the trace output data set and to perform a dump. This also causes zFS to attempt an internal restart.
Determining the XCF protocol interface level

Beginning with z/OS V2R1, zFS uses the extended directory XCF communications protocol and runs with sysplex=filesys. This change requires toleration support on V1R12 and V1R13. For more information, see [z/OS Migration](#).

Message IOEZ00617I is issued during zFS initialization to indicate whether zFS is running sysplex-aware on a file system basis (referred to as sysplex filesys), sysplex-aware for all read/write file systems (referred to as sysplex file-support), or neither (referred to as sysplex admin-only). It also indicates the zFS interface level that is being used:

4 One of the following:
   - The z/OS V2R1 level (with XCF protocol extended directory)
   - The z/OS V1R13 level (with XCF enhanced connect protocol)
3 The z/OS V1R12 level
2 The V1R12-compatible level used by z/OS V1R9 and z/OS V1R10

Saving initialization messages in a data set

The IOEFSPRM msg_output_dsn option specifies the name of a data set that contains output messages that come from the zFS PFS during zFS initialization. This option might be helpful for debugging because the data set can be sent to IBM service if needed. The msg_output_dsn option is optional. If it is not specified, zFS PFS messages go only to the system log. If it is specified, the data set should be preallocated as a sequential data set with a RECFM=VB and LRECL=248 and should be large enough to contain all zFS PFS initialization messages between restarts. The space used depends on how many zFS initialization messages are issued. A suggested primary allocation is two cylinders with a secondary allocation of two cylinders. If the data set fills up, no more messages are written to the data set. (They still go to the system log.) After zFS restarts, the message output data set is overwritten.

Determining service levels

You can determine the service level of the zFS physical file system by examining the messages that occur on the operator's console when zFS initializes.

IOEZ00559I zFS kernel: Initializing z/OS File System
Version 02.01.00 Service Level 0000000 - HZFS410.
Created on Wed Mar 20 16:05:20 EDT 2013.
Address space asid x45

Alternatively, you can issue the MODIFY ZFS,QUERY,LEVEL operator command and look for the following message:

IOEZ00639I zFS kernel: z/OS File System
Version 02.01.00 Service Level 0000000 - HZFS410.
Created on Wed Mar 20 16:05:20 EDT 2013.
sysplex(filesys,norwshare) interface(4)

In a z/OS V1R13 or later shared file system environment, the sysplex level is (filesys,norwshare) or (filesys,rwshare), depending on the sysplex_filesys_sharemode and the interface is (4).

In addition, you can determine the service level of the zfsadm command by issuing the -level option of the zfsadm command. For example:

zfsadm -level

IOEZ00020I zfsadm: z/OS File System
Version 02.01.00 Service Level 0000000 - HZFS410.
Created on Wed Mar 20 16:05:20 EDT 2013.
Understanding namespace validation and correction

zFS provides namespace validation and correction in a shared file system environment. First, it is important to understand the concept of a namespace. zFS communicates between sysplex members using XCF protocols. The zFS XCF protocol exchanges information among members about zFS ownership and other attributes of zFS mounted file systems. This information, which is kept in the memory of each zFS member, is called the zFS namespace. If zFS members do not agree on the zFS owner of each file system, there might be problems that require a zFS restart or an IPL to recover.

zFS namespace validation is invoked in one of four ways:
- When an administration command experiences an XCF message timeout
- Automatically at zFS initialization
- Automatically when zFS detects a problem that might be because of a namespace inconsistency
- Explicitly using the MODIFY ZFS,NSVALIDATE operator command.

zFS namespace validation compares the information that is stored in each zFS member. If zFS validation detects an inconsistency, one or more messages can occur (for example, IOEZ00612I) and zFS attempts to correct the inconsistency, using one of the following actions:
- Updating the inconsistent information
- Automatically remounting a file system
- Internally restarting zFS on one or more members.

The corrective action is disruptive and might cause one or more applications to receive I/O errors and display messages IOEZ00618E through IOEZ00637E. In addition, zFS might take SVC dumps when it detects a name inconsistency; therefore, do not issue the MODIFY ZFS,DUMP,ALL command.

Each zFS only keeps track of file systems that are locally mounted. z/OS UNIX locally mounts file systems on systems where the mount was issued (or directed to through the SYSNAME parameter), and for sysplex-aware file systems, on other systems. z/OS UNIX keeps mount information that is hardened in the couple data set. In addition, zFS keeps track of zFS ownership by using cross system ENQ. The zFS owner of an aggregate always has an exclusive ENQ with a qname of SYSZIOEZ and an rname of IOEZLT.aggregatename. In this way, zFS hardens zFS ownership information in an independent repository. When an inconsistency is detected in the zFS namespace information between zFS members, this hardened information can be queried to determine how to automatically correct the inconsistency.

Tip: You can use the DISPLAY GRS,RES=(SYSZIOEZ,*) operator command to display zFS ENQs. For RNAME explanations and use, see the topic on Serialization summary in z/OS MVS Diagnosis: Reference.

Requirement: Because of the enhanced directory XCF protocol between zFS members, and because zFS always runs as sysplex=filesys, two additional steps must be performed on z/OS V1R12 to work with zFS on z/OS V1R13 and later releases:
- You must apply toleration APAR OA39466.
- zFS on V1R12 must run at sysplex=filesys to allow zFS on z/OS V1R13 and later releases to enter the shared file system environment.
Understanding delays and hangs in zFS using the zFS hang detector

The zFS hang detector automatically monitors the current location of the various tasks processing in zFS. At a set interval, the hang detector thread wakes up and scans the current user requests that have been called into zFS. The hang detector processes this list of tasks and notes various pieces of information to determine the location of the task. When the hang detector determines that a task has remained in the same location for a predefined period of time, it attempts to determine why it is not making progress. This might cause zFS messages or dumps. Certain zFS messages can remain on the screen while the delay continues. If subsequently, the hang detector recognizes that this task has finally progressed, it DOMs the zFS message (removes it from the console). If the zFS message is removed, it means that the delay has cleared and was just a slowdown because of a stressful workload or some other issue. In this case, you can discard any zFS dumps that occur because of this delay.

Several zFS messages warn of potential problems in the zFS address space that have to do with delays. If zFS determines there is a true deadlock, zFS initiates dumps of all systems. The system that detected the deadlock stops and restarts zFS to clear the deadlock. Some delays involve only a single system; other delays in a shared file system environment can involve other systems and XCF communications.

IOEZ00xxxI zFS messages are issued by the zFS hang detector and generally remain on the console until the situation is resolved. Resolution occurs when:

- The delayed task completes without any external correction. This is a slowdown and not a hang. Discard any zFS system dumps.
- The delayed task is cancelled or the request is timed out. In these cases, you should supply any system dump taken by zFS to IBM service for diagnosis.

For delays, zFS issues several messages to attempt to diagnose what might be involved in the delay. A delay might occur when:

- zFS invokes another component (such as allocation, open/close, or global resource serialization). In this case, zFS issues message IOEZ00604I or IOEZ00660I to recommend that you use the other component's diagnosis material to determine the cause of the delay. zFS does not produce a dump.
- There is heavy system activity with higher priority tasks delaying lower priority tasks or a delay in another system service not covered by message IOEZ00604I. In this case, zFS issues message IOEZ00605I, but does not produce a dump.

Hangs and delays in shared file system environment

When there is an XCF communication delay, the zFS hang detector can determine:

- The other system never received the XCF message. zFS issues message IOEZ00591I.
- The other system received the XCF message, but it is not making any progress on the other system or zFS cannot determine its status. zFS issues message IOEZ00547I.
- The other system received the XCF message but the progress is very slow or long running. zFS issues message IOEZ00661I.
The other system has processed the XCF message and sent a response back, but zFS has not received the response. zFS issues message IOEZ00592I.

In these cases, no system dump is issued by zFS. Use the message information that refers to the systems that are not responding and determine the status of those systems. There might also be messages on the other systems that indicate the real problem. (Typically, each system issues its own messages when there is a problem.) There are time-outs on each XCF message. It’s prudent to wait to see if a request timing out resolves the hang. If a request times out, the request will fail.

zFS also determines how long remote requests can take by supplying a timeout value to XCF (approximately 10 to 15 minutes). XCF monitors the request and if it takes longer than the timeout value, XCF indicates to zFS that the request timed out. In this case, zFS issues message IOEZ00658E or IOEZ00659E and fails the request. The message indicates an aggregate name if the timeout can be associated with an aggregate. The administrator should use the information in the message that refers to the system that is not responding and determine the status of that system. You might see zFS hang detector messages and the operation might not have run on the target system.

**Steps for diagnosing and resolving a zFS hang**

**About this task**

Perform the following steps when a hang condition is suspected.

**Procedure**

1. Continually monitor for the following messages:

   - **IOEZ00524I**
     
     zFS has a potentially hanging thread caused by: UserList, where: UserList is a list of address space IDs and TCB addresses causing the hang.

   - **IOEZ00547I**
     
     zFS has a potentially hanging XCF request on systems: Systemnames, where: Systemnames is the list of system names.

   To start investigating, if in a sysplex file sharing environment check for message IOEZ00547I (hanging XCF request), which can indicate an XCF issue. If you see this message:
   a. Check the status of XCF on each system in the sysplex.
   b. Check for any outstanding message that might need a response to determine whether a system is leaving the sysplex or not (for example, IXC402D). The wait for a response to the message might appear to be a zFS hang.

   If there is no apparent problem with XCF, continue diagnosis and resolution of the hang by looking for the following messages in syslog or on the operator console. Check each system in the sysplex if applicable.

   - **IOEZ00604I or IOEZ00660I**
     
     The delay is outside of zFS. zFS called the identified system service and is waiting for a response. Investigate the identified system service. The problem is likely not with zFS.

   - **IOEZ00605I**
     
     The delay is either in zFS or in a system service that zFS did not specifically identify in message IOEZ00604I. zFS cannot determine whether there is a hang, a slowdown, or some other system problem. To take action, look for other symptoms. For example, if you see messages
about components that are using a significant amount of auxiliary storage, resolve the auxiliary storage shortage. If the message persists, continue to the next step.

2. Enter the MODIFY ZFS,QUERY,THREADS command to determine whether any zFS threads are hanging and why.

   The type and amount of information displayed as a result of this command is for internal use and can vary between releases or service levels. For an example, see [Figure 43 on page 108]

3. Enter the DISPLAY A,ZFS command to determine the zFS ASID.

4. Enter MODIFY ZFS,QUERY,THREADS at one to two minute intervals for six minutes.

5. Check the output for any user tasks (tasks that do not show the zFS ASID) that are repeatedly in the same state during the time you requested MODIFY ZFS,QUERY,THREADS. If there is a hang, the task that is hanging persists unchanged over the course of this time span. If the information is different each time, there is no hang.

6. If message IOEZ00581E is highlighted in white on the console, there are or recently were quiesced zFS aggregates. Verify that no zFS aggregates are in the QUIESCED state by checking their status using the zfsadm lsaggr or zfsadm aggrinfo -long command. For example, quiesced aggregates are displayed as follows:

   ```
   DCESVPI:/home/susvpi/> zfsadm lsaggr
   IOEZ00106I A total of 1 aggregates are attached
   SUSVPI.HIGHRISK.TEST DCESVPI R/W QUIESCE
   DCESVPI:/home/susvpi/> zfsadm aggrinfo
   IOEZ001701I A total of 1 aggregates are attached.
   SUSVPI.HIGHRISK.TEST (R/W COMP QUIESCED): 35582 K free out of total 36000
   DCESVPI:/home/susvpi/>
   ```

   or

   ```
   DCESVPI:/home/susvpi/> zfsadm aggrinfo susvpi.highrisk.test1.zfs -long
   SUSVPI.HIGHRISK.TEST1.ZFS (R/W COMP QUIESCED): 50333 K free out of total 72000
   version 1.4
   auditfid 00000000 00000000 0000
   6289 free 8k blocks; 21 free 1K fragments
   720 K log file; 40 K filesystem table
   16 K bitmap file
   Quiesced by job SUSVPI5 on system DCESVPI on Tue Jan 3 13:36:37 2013
   DCESVPI:/home/susvpi/>:
   ```

   If the hang condition prevents you from issuing shell commands, you can also issue the MODIFY ZFS,QUERY,FILE,ALL command to determine whether any file systems are quiesced. As [Figure 35 on page 76] shows, a quiesced file system is identified by a “Q” in the flg column.

   Resolve the QUIESCED state before continuing to the next step. The hang condition message can remain on the console for up to a minute after the aggregate is unquiesced.

   Message IOEZ00581E appears on the zFS owning systems that contain at least one zFS aggregate that is quiesced. There is a delay between the time that the aggregate is quiesced and the time that the message appears. Typically, this time delay is about 30 seconds. You can control this time delay by using the IOEFSPRM QUIESCE_MESSAGE_DELAY option. This option allows you to specify that the delay should be longer than 30 seconds before the IOEZ00581E message is first displayed. When there are no quiesced zFS aggregates on the system, this message is removed from the console. There is also a delay between the time that the last aggregate is unquiesced and the
time that the message is removed from the console. This message is handled by a thread that wakes up every 30 seconds and checks for any quiesced aggregates owned by this system. It is possible for an aggregate to be quiesced and unquiesced in the 30-second sleep window of the thread and not produce a quiesce message. This message remains if one aggregate is unquiesced and another is quiesced within the 30-second sleep window.

7. Check whether any user tasks are hung, focusing on the tasks identified by message IOEZ00524I or message IOEZ00660I. User tasks do not have the same address space identifier (ASID) as the zFS address space. One or more threads consistently at the same location might indicate a hang (for example, Recov, TCB, ASID Stack, Routine, State). The threads in the zFS address space with the zFS ASID (for example, xcf_server) are typically waiting for work. It is typical for the routine these threads are waiting in to have the same name as the entry routine. For an example, see Figure 43 on page 108.

If successive iterations of the MODIFY ZFS,QUERY,THREADS command show that the STK/Recov, TCB, ASID, Routine, and State for a thread are constant, it is probable that this thread is hung.
8. IBM Support must have dumps of zFS, OMVS and the OMVS data spaces and also possibly the user address space identified on any preceding IOEZ00605 for problem resolution. Obtain and save SYSLOG and dumps of zFS, OMVS and the OMVS data spaces , and the user ASID using JOBNAME=(OMVS,ZFS,user_jobname),DSPNAME=('OMVS'.*) in your reply to the DUMP command. If you are running in a sysplex and zFS is running on other systems in the sysplex, dump all the systems in the sysplex where zFS is running, dumping zFS, OMVS and OMVS data spaces. The following is an example of the DUMP command:

DUMP COMM=(zfs hang)
R x,JOBNAME=(OMVS,ZFS),SDATA=(RGN,LPA,QSA,LSQA,PSA,CSA,GRSQ,TRT,SUM,COUPLE),JOBNAME=(OMVS,ZFS,user_jobname),DSPNAME=('OMVS'.*),END

Rule: You must capture dumps for IBM Support before taking any recovery actions (HANGBREAK, CANCEL, ABORT).
9. If you know which user task is hung (for example, returned in IOEZ00524I or determined to be hung after review of the output from repeated MODIFY ZFS,QUERY,THREADS,OLDEST commands), consider entering the CANCEL or STOP command to clear that task from the system.

10. Finally, if the previous steps do not clear the hang, issue the MODIFY ZFS,ABORT command to initiate a zFS internal restart.

   An internal restart causes the zFS kernel (IOEFSKN) to end and then restart, under control of the zFS controller task (IOEFSCM). The zFS address space does not end and the z/OS UNIX mount tree is preserved. During the internal restart, requests that are already in the zFS address space fail and new requests are suspended. File systems owned by zFS on the system that is doing the internal restart become temporarily unowned. These file systems are taken over by other zFS systems (or by the zFS system doing the internal restart when it completes the internal restart). When the internal restart is complete, the suspended new requests resume.

   If you question the hang condition or if the MODIFY ZFS,ABORT command does not resolve the situation, contact IBM Support and provide all the dumps and SYSLOG information.

Identifying storage shortages in zFS

When zFS can no longer obtain sufficient storage to complete a request, it issues message IOEZ00188A, possibly creates a dump, and restarts. If you see message IOEZ00188A before zFS initialization is complete (before message IOEZ00055I), either increase the REGION size in the ZFS PROC or decrease some cache sizes in the IOEFSRM configuration file.

   In addition, the zFS hang detector periodically checks a warning limit and a critical limit. When it reaches the warning limit, message IOEZ00662I displays and remains on the console until the situation is resolved, or until the critical limit is reached. If the critical limit is reached, message IOEZ00663I displays and remains on the console until storage usage goes below the critical limit to the warning limit, and then message IOEZ00662I displays again. See “STOR” on page 89 for more information about how to determine the amount of storage being used in the zFS address space.

A zFS storage shortage can be caused by the number of active vnodes in use in zFS. You can query the number of held vnodes using either the MODIFY ZFS,QUERY,LFS system command or the `zfsadm query -vnodecache` command. You can also query the current sizes of the zFS caches in the zFS address space using the `zfsadm configquery` command with its cache size parameters, such as `-tran_cache_size` and `-vnode_cache_size`. For example, `zfsadm configquery -meta_cache_size` returns the metadata cache size. When zFS is running in a shared file system environment, you can query the client reply storage using `zfsadm configquery -client_reply_storage`. You can also determine cache sizes by using the MODIFY ZFS,QUERY,STORAGE command. Decreasing one or more cache sizes might relieve the zFS storage shortage.

**Restriction:** Changing the size of a cache can cause delays. Try to do it during low activity periods.

In general, if you see a return code of 132 (ENOMEM), zFS is short on storage; take steps to reduce zFS storage usage. When storage shortages become critical, you can also see 157 (EMVSERR) and mounts might begin to fail.
Diagnosing disabled aggregates

If zFS detects a problem on an aggregate that is mounted read/write, zFS attempts to isolate the failure. As a result, zFS might mark an aggregate unavailable and issue message IOEZ00422E, as shown in the following example.

IOEZ00422E Aggregate PLEX.JMS.AGGR001.LDS0001 disabled

In addition, a dump and possibly zFS trace information might be generated. You can contact IBM service and provide the dump and the trace and any other information that is useful for diagnosing the problem (for example, what was running on the system when the problem occurred).

When an aggregate is disabled, applications cannot read from, or write to, the aggregate. Other aggregates that are not involved in the failure remain available. However, the disabled aggregate is unavailable for reading and writing until it is unmounted and mounted. Beginning with z/OS V1R13, if the disabled aggregate is zFS owned on a zFS V1R13 or later system, zFS attempts to automatically re-enable the disabled aggregate and make it available again for use.

- zFS attempts an internal remount samemode on the zFS-owning z/OS V1R13 or later system in the following situations:
  - It is in a non-shared file system environment
  - The file system is non-sysplex aware
  - The file system is sysplex-aware, but no other z/OS V1R13 or later system in the shared file system environment can take it over
- Alternatively, in a shared file system environment where the file system is sysplex-aware, the zFS owning system requests that another system that is running z/OS V1R11 or later take over the aggregate.

The preceding re-enablement actions (aggregate movement or internal remount samemode) are taken only if the file system became disabled due to an internal zFS error or a corruption.

Note that although the aggregate is disabled, z/OS UNIX System Services continues to display the aggregate mounted as R/W. To determine whether the aggregate has been marked as disabled, use the zfsadm lsaggr command or the zfsadm aggrinfo command.

An aggregate that has been disabled might be corrupted, even if it has been disabled and remounted. To be sure that the aggregate is internally consistent, run the ioefsutl salvage batch utility against the aggregate that was disabled, to repair any corruption, and prevent loss of data. See “ioefsutl” on page 134 for more information.

Handling disabled aggregates

An aggregate can become disabled for many reasons, such as:
- An I/O error or failure of a DASD device
- Loss of connectivity to a DASD device
- An internal zFS error
- Permanent corruption of the aggregate

The procedure to re-enable the aggregate varies, depending on the release level of zFS running in the shared file system environment:
- When some systems are running z/OS V1R13 or later, see “Disabled aggregates when some systems are on z/OS V1R13 or later” on page 111.
When no systems are running z/OS V1R13 or later, see "Disabled aggregates when there are no z/OS V1R13 or later systems."

**Disabled aggregates when some systems are on z/OS V1R13 or later**

The following information applies when some systems are on z/OS V1R13 or later. If the compatibility mode aggregate becomes disabled and the zFS owning system is running z/OS V1R13 or later, zFS attempts to automatically re-enable the disabled aggregate. It either requests that another system running z/OS V1R11 or later in the shared file system environment take over the aggregate (if it is sysplex-aware) or it attempts an internal remount samemode. This action should recover the aggregate and it will no longer be disabled. If the compatibility mode aggregate is not zFS owned on a z/OS V1R13 or later system, follow the procedure described in "Disabled aggregates when there are no z/OS V1R13 or later systems."

As a general rule, an aggregate that has become disabled (unless it was due to a planned activity, such as a vary offline of a device) should be salvaged by using the `ioefsutl salvage` utility at your earliest convenience. Because zFS has detected a problem, there is a chance that the file system is corrupted, even if it has been successfully re-enabled. If the file system can be taken offline (unmounted) immediately or at a regularly-scheduled time, you should do so and run salvager. However, if the file system is a critical production file system that cannot be easily unmounted, you will have to use judgement considering the inconvenience of unmounting the file system against the risk of continuing to use a file system that may possibly be corrupted. When the file system is backed up according to your installation’s regular schedule, you might be backing up a corrupted file system; if this continues, you might lose any previous backed-up versions of the file system that were not corrupted. In this case, you may want to arrange to salvage the first backup copy of the file system after it was disabled and re-enabled.

To run the `ioefsutl salvage` utility, you must first unmount the aggregate. Note that the z/OS UNIX shell `unmount` command (`/usr/sbin/unmount`) may query the status of the file system before unmounting it. Because the file system is disabled, this query will fail which, in turn, may cause the entire unmount to fail. Therefore, you may need to use the TSO/E UNMOUNT command or the operator MODIFY BPXOINIT,FILESYS=UNMOUNT,FILESYSTEM=filesysname command to unmount the disabled file system. If you do not unmount before running `ioefsutl salvage` the system issues messages, such as the following one:

```
IKJ56225I DATA SET PLEX.JMS.AGGR001.LDS0001 ALREADY IN USE, TRY LATER+
IKJ56225I DATA SET IS ALLOCATED TO ANOTHER JOB OR USER
IOEZ0003E While opening minor device 1, could not open dataset
PLEX.JMS.AGGR001.LDS0001.
```

After you run the `ioefsutl salvage` utility and are satisfied that the aggregate is in a consistent state, mount the aggregate again.

**Disabled aggregates when there are no z/OS V1R13 or later systems**

When there are no z/OS V1R13 or later systems in the shared file system environment, if the compatibility mode aggregate becomes disabled because of an I/O error or corruption, unmount the file system and any other file systems that are mounted below the disabled aggregate. Note that the z/OS UNIX shell `unmount` command (`/usr/sbin/unmount`) might query the status of the file system before unmounting it. Because the file system is disabled, this query will fail which, in turn, might cause the entire unmount to fail. Therefore, you might need to use the
TSO/E UNMOUNT command or the operator command MODIFY BPXOINIT,FILESYS=UNMOUNT,FILESYSTEM=filesysname to unmount the disabled file system. If you do not unmount before running ioeagslv, the system issues messages similar to those shown in “Disabled aggregates when some systems are on z/OS V1R13 or later” on page 111.

Run the ioeagslv utility to ensure that the aggregate is internally consistent. When you are satisfied that the aggregate is in a consistent state, mount the aggregate.

If you choose not to run ioeagslv, you still need to unmount and mount the aggregate so that it will not remain disabled. If there are file systems that are mounted below the disabled aggregate, you can use the remount capability of z/OS UNIX to avoid unmounting those lower file systems. Remount allows you to change a mounted file system from read-only to read/write, from read/write to read-only, or keep the same mode (read to read and write to write) without affecting lower mounted file systems.

For example, if PLEX.JMS.AGGR001.LDS0001 is mounted read/write at /zfsmnt1, you can keep the mount read/write, using one of the following methods:

- Enter the following TSO/E UNMOUNT command:
  UNMOUNT FILESYSTEM('PLEX.JMS.AGGR001.LDS0001') REMOUNT(SAMEMODE)

- Enter the following z/OS UNIX chmount command:
  /usr/sbin/chmount -s /zfsmnt1

- Use the Work with Mounted File Systems ISHELL panel:

You can use the df -v command to determine if your file system is mounted and whether it is mounted read-only or read/write. For more information about these commands, see the topics on UNMOUNT and chmount in z/OS UNIX System Services Command Reference.
Chapter 9. Overview of the zFS audit identifier

An auditid is a 16-byte value that is associated with each z/OS UNIX file or directory. The auditid identifies a z/OS UNIX file or directory in an SMF audit record or in certain authorization failure messages (for example, RACF message ICH408I). An auditid appears in Type 80 SMF records and in the output of certain z/OS UNIX APIs (for example, stat). zFS allows the administrator to specify whether zFS uses a more unique auditid for a zFS file or directory, or uses the non-unique, standard auditid.

Tip: The auditid tool can display a file path name if you know the auditid. The tool works only for a unique auditid; it does not work for a standard auditid. It is not supported and is available on the z/OS UNIX System Services Tools and Toys web page (http://www.ibm.com/systems/z/os/zos/features/unix/bpxa1ty2.html).

Figure 44 shows the format of the unique zFS auditid, the standard zFS auditid, and the HFS auditid.

Together, the i-node and unique identifier identify the file or directory within a file system. The remainder of the auditid identifies the file system. The i-node is a slot number that identifies an existing file or directory, but it is reused when a file or directory is deleted. When that same i-node slot is used for a different file or directory, the uniquifier is incremented so that the combination of the i-node and uniquifier is unique. When the uniquifier is two bytes, they are the low order bytes (the bytes that change most often) of the four-byte uniquifier. In the unique zFS auditid, the file system part of the auditid is known as the auditfid. The VOLSER is the volume serial of the volume that contains the first extent of the zFS aggregate data set. The CCHH is the CCHH of the first extent of the zFS aggregate data set.

The auditfid in the zFS aggregate controls the type of auditid zFS uses: unique auditid or less unique auditid (auditfid of binary zeros). Typically, a zFS aggregate contains a zero auditfid, but you can take steps to store a unique zFS auditfid, which subsequently causes zFS to generate a unique format auditid for each file or directory in the aggregate.
There are three ways to control the zFS auditfid that is stored in the aggregate, which thereby controls the format of the zFS auditid for files and directories that are contained in the aggregate:

- When formatting an aggregate, you get a unique auditfid by default (that is, if you do not specify -nonewauditfid). This is true for the IOEAGFMT batch utility and the `zfsadm format` command. If you specify -nonewauditfid, the aggregate has the standard auditfid (binary zeros). The IOEFSUTL format always provides a unique auditfid.

- You can optionally specify a zFS configuration option (convert_auditfid=on) in the IOEFSPRM file to control whether the aggregate's auditfid is converted from a standard format auditfid to a unique auditfid when a zFS file system is mounted. If you specify on, zFS converts the standard auditfid to the unique auditfid on the read/write mount (attach) of the aggregate. You can also specify the convert_auditfid configuration option using the `zfsadm config -convert_auditfid` option and query using the `zfsadm configquery -convert_auditfid` option. The default for convert_auditfid is on.

- You can explicitly set an aggregate's auditfid to a unique auditfid using the `zfsadm setauditfid` command.

---

**Enabling the zFS auditid**

To enable the unique auditid, start by following scenario 2 with some new aggregates to verify that it does not cause problems for your installation. Then, use scenario 3 to convert the rest of the aggregates. The next time the aggregates are mounted, they have a unique auditfid.

**Scenarios:**

1. You want all your aggregates to have the unique auditfid (and therefore, all auditids) use the new method:
   a. Do nothing. The default is convert_auditfid=on in your IOEPRMxx configuration file and new aggregates get unique auditfids by default.

   **Result:** Any existing aggregates are converted to the unique auditfid the next time they are mounted (attached). Newly formatted aggregates using IOEAGFMT, or `zfsadm format` get unique auditfids by default. IOEFSUTL format always creates unique auditfids.

2. You want your new aggregates to have the unique auditfid and your existing aggregates to remain with the standard auditfid:
   a. Specify convert_auditfid=off in your IOEPRMxx configuration file.
   b. Specify (or default to) -newauditfid when you format new aggregates using IOEAGFMT or `zfsadm format`. Use IOEFSUTL to format new aggregates.

   **Result:** Old aggregates are not converted to unique auditfids when you mount (attach), but new aggregates have the unique auditfids.

3. You want all your aggregates to remain with the standard auditfid (and therefore all auditids have the standard format):
   a. Specify convert_auditfid=off in your IOEPRMxx configuration file and specify -nonewauditfid when you use IOEAGFMT or `zfsadm format` to format new aggregates. Do not use IOEFSUTL format to format new aggregates.

   **Result:** Any existing aggregates are converted to the unique auditfid the next time they are mounted (attached). When you format new aggregates and specify the -newauditfid option, the aggregates have the unique auditfid.
**Guideline:** New aggregates formatted with ISHELL, automount allocany, allocuser, or the BPXWH2Z utility will not have unique auditfids after they are formatted. However, they will be converted to unique auditfids by default the first time they are mounted unless you specify `convert_auditfid=off` in your IOEPRMxx configuration file or specify `zfsadm config -convert_auditfid off`.

If a zFS aggregate is moved to another DASD location, the auditfid remains the same, unless you change it using the `zfsadm setauditfid -force` command. This is a trade-off between changing the auditfid, which causes auditids for the same file to be generated differently, versus not changing the auditfid, which causes auditids to remain the same but with the possibility that another zFS aggregate might get allocated with the first extent exactly in the place (and on the same volume) as the moved aggregate was located. This means that two different zFS files/directories might have the same auditid.

Even though the zFS auditid format is described, the internal contents of an auditid might not match exactly as stated. The VOLSER might not match the VOLSER of the volume containing the first extent because of moving the aggregate. The main use should be as an opaque number (that is, you should only use it to compare for equality of the whole auditid against another auditid).

Use the following algorithm to help distinguish between the unique auditid, the standard zFS auditid, and HFS auditid (which does not depend on the internal contents of the new zFS auditid):

```plaintext
If the last eight bytes of the auditid are binary zero, the auditid is zFS standard format
Else, if the first byte of the auditid is X'01', the auditid is an HFS format
Else, the auditid is the unique zFS format
```
Part 2. zFS administration reference

This part of the document contains reference information for zFS, and includes the following topics:

- Chapter 10, “z/OS system commands,” on page 119
- Chapter 11, “zFS commands,” on page 125
- Chapter 12, “The zFS configuration options file (IOEPRMxx or IOESPRM),” on page 193
- Chapter 13, “zFS application programming interfaces,” on page 207.
Chapter 10. z/OS system commands

This section introduces you to the following z/OS system commands:

- MODIFY, a system command that enables you to query internal counters and values. It also allows you to initiate or gather debugging information.
- SETOMVS RESET, a system command that starts the zFS Physical File System (PFS) if it has not been started at IPL, or if the PFS has been stopped and the BPXF032D message has been responded to with a reply of i.

These commands can be run from the console or from System Display and Search Facility (SDSF).

modify zfs process

Purpose

Enables you to query internal zFS counters and values. They are displayed on the system log. It also allows you to initiate or gather debugging information. The zFS PFS must be running to use this command.

Format

You can use any of the following formats for this command.

```
modify procsname,query,{level | settings | threads,{allwait | oldest}} | status | 
{(kn | vm | lfs | lock | storage | file | stkm | ctck | svi | iobydasd | dataset | all)} 
modify procsname,reset,{kn | vm | lfs | lock | storage | file | stkm | ctck | svi | iobydasd | 
        dataset | all} 
modify procsname,trace,{reset | print} 
modify procsname,abort 
modify procsname,dump 
modify procsname,hangbreak 
modify procsname,unquiesce,aggregate_name 
modify procsname,nsvalidate[,print] 
```

Parameters

`procsname`

The name of the zFS PFS PROC. The default procsname is ZFS.

`command`

The action that is performed on the zFS PFS. This parameter can have one of the following values:

`query`  Displays zFS counters or values.

`level`  Displays the zFS level for the zFS physical file system kernel. When running in a shared file system environment, `level` also displays the zFS sysplex level and the zFS XCF communication interface level (1, 2, 3 or 4). The zFS sysplex level is controlled by the IOEFSRPM sysplex.
modify zfs process

configuration option. When the sysplex level is filesys, the default mount PARM (NORWSHARE or RWSHARE) is also displayed. (As of z/OS V1R13, zFS always runs with sysplex=filesys.) For an example and more information, see "Determining service levels" on page 102.

settings
Displays the zFS configuration settings, which are based on the IOEFSprm file and defaults.

threads[,allwait | oldest ]
Displays the threads being monitored by the zFS hang detector. To display all zFS threads, use the modify zfs,query,threads,allwait command. The time of day values are shown in Greenwich mean time (GMT). To display the oldest thread of each system, use the modify zfs,query,threads,oldest command.

status
Displays zFS internal restart information.

<report>
One of the following report options; for details about these reports, see to "Monitoring zFS performance” on page 73.

kn
Displays the calls made to zFS from z/OS UNIX.

vm
Displays the user file cache including cache hit ratios, I/O rates and storage usage.

lfs
Displays the file system statistics including the performance of the zFS metadata caches, the vnode cache and the aggregate I/O statistics.

log
Displays the log statistics.

lock
Displays the lock contention values.

storage
Displays the zFS storage values.

file
Displays the requests per zFS file system and aggregate.

stkm
Displays the current server token manager (STKM) statistics.

ctkc
Displays the client calls to other systems. Output is only displayed when the zFS ctkc component on this system has sent a message to another system.

svi
Displays the calls from other systems to this server through the server vnode interface (SVI) component. Output is only displayed when the zFS svi component on this system has received a message from another system.

iobydasd
Displays the DASD that is currently attached by volume.

dataset
Displays zFS statistics about file systems.

all
Displays all the zFS counters.
modify zfs process

reset  Resets zFS counters and consists of the following options:
  kn    Resets the calls made to zFS from z/OS UNIX.
  vm    Resets the user file cache including cache hit ratios, I/O rates and storage usage.
  lfs   Resets the file system statistics including the performance of the zFS metadata caches, the vnode cache and the aggregate I/O statistics.
  log   Resets the log statistics.
  lock  Resets the lock contention values.
  storage
        Resets the zFS storage counters.
  file  Resets the requests for zFS file system and aggregate.
  stkm  Resets the server token manager (STKM) statistics.
  ctkc  Resets the client call statistics.
  svi   Resets the received calls from other systems statistics.
  iobydasd
        Resets the count of the DASD that is currently attached by volume.
  dataset
        Reset the zFS statistics about file systems.
  all   Resets all the zFS counters to zero.
trace  Resets or prints the internal zFS trace table.
  reset  Resets the internal (wrap around) trace table to empty.
  print  Formats and sends the current trace table to the data set specified in the IOEFSPRM file trace_dsn entry. This data set must be pre-allocated as a PDSE with RECFM VB and LRECL 133. It must be large enough to hold the formatted trace table. See Chapter 8, "Performance and debugging," on page 69 for more information about the trace output data set.
  abort  Causes zFS to dump and then perform an internal restart. The internal trace table is also printed to the data set specified in the IOEFSPRM file trace_dsn entry.
  dump  Causes the zFS PFS to dump and to print the internal trace table to the data set specified in the IOEFSPRM file trace_dsn entry.
  hangbreak
        Causes a zFS internal restart; this produces the same result as issuing a modify zfs, abort command.
  unquiesce
        Causes a quiesced aggregate to become unquiesced. Only locally attached aggregates can be unquiesced using the modify unquiesce command. You must issue this command on the system that is the zFS owner of the aggregate. Use the z/OS UNIX zfsadm lsaggr command to determine which system is the zFS owner of the aggregate.
modify zfs process

nsvalidate

Initiates the zFS namespace validation on the system where the command is entered. The modify nsvalidate command should only be used in a shared file system environment; typically, it is only used as a part of a recovery procedure when a problem with zFS is suspected. If the command finds an inconsistency, it might cause zFS to abort and internally restart the zFS address space on one or more systems to correct the zFS namespace inconsistency. The modify nsvalidate command consists of the following option:

print The optional print parameter displays additional name space information that is obtained after validation.

Usage

The modify zfs command is used to display zFS counters or values and to initiate or gather debugging information. You cannot issue modify zfs commands during a zFS internal restart.

Privilege required

This command is a z/OS system command.

Examples

The following example queries all the zFS counters:
modify zfs,query,all

The following example resets the zFS storage counters:
modify zfs,reset,storage

The following example formats and sends the trace table to the data set specified in the IOEFSPRM file trace_dsn entry:
modify zfs,trace,print

The following example causes the zFS PFS to execute an internal restart:
modify zfs,abort

Related information

File:
IOEFSPRM

For details on stopping zFS, see the topic on Recycling z/OS UNIX System Services in z/OS MVS System Commands.

setomvs reset

Purpose

Can be used to start the zFS PFS if it has not been started at IPL. It can also be used to redefine it if it has been terminated by replying i to the BPXF032D operator message (after stopping the zFS PFS).
setomvs reset

Format

```
setomvs reset={xx}
```

Parameters

```
x
```
x The suffix of a BPXPRMxx member of PARMLIB that contains the FILESYSTYPE statement for the zFS PFS.

Usage

The `setomvs reset` command can be used to start the zFS PFS.

Privilege required

This command is a z/OS system command.

Examples

The following command starts the zFS Physical File System if the BPXPRMSS member of the PARMLIB contains the zFS FILESYSTYPE statement:

```
setomvs reset=(ss)
```

Related information

File:

IOEFSPRM

In z/OS V1R7 and above, the SETOMVS command also processes zFS FILESYSTYPE statements. For more information, see `SETOMVS command` in `z/OS MVS System Commands`.
setomvs reset
Chapter 11. zFS commands

This section provides a description of zFS commands and batch utilities. In the options section for each command, options are described in alphabetic order to make them easier to locate; this does not reflect the format of the command. The formats are presented the same as on your system.

In addition to displaying z/OS UNIX reason codes, the z/OS UNIX shell command, `bpxmtext`, also displays the text and action of zFS reason codes (EFxxnnnn) returned from the kernel. zFS does not use the `xx` part of the reason code to display a module name. It always displays zFS. If you only know the `nnnn` part of the zFS reason code, you can use `EF00nnnn` as the reason code. The date and time returned with the zFS reason code matches the date and time returned from the zFS kernel (displayed with operator command `MODIFY ZFS,QUERY,LEVEL`). For additional information about the `bpxmtext` command, see z/OS UNIX System Services Command Reference.

Note: The `bpxmtext` command is not valid for zFS abend reason codes (EAxxnnnn).

You can use the `man` command to view the descriptions of zFS command manual pages. To use man pages, enter `man` followed by the command information you want to display. You must enter the `zfsadm` command suite entries as one word. Table 9 shows examples of the zFS man commands.

<table>
<thead>
<tr>
<th>zFS command</th>
<th>man command</th>
</tr>
</thead>
<tbody>
<tr>
<td>ioeofsutl salvage</td>
<td>man ioeofsutlsalvage</td>
</tr>
<tr>
<td>ioeagfmt</td>
<td>man ioeagfmt</td>
</tr>
<tr>
<td>mount</td>
<td>man zfsmount</td>
</tr>
<tr>
<td>zfsadm aggrinfo</td>
<td>man zfsadmaggrinfo</td>
</tr>
<tr>
<td>zfsadm query</td>
<td>man zfsadmqery</td>
</tr>
</tbody>
</table>

Table 9. zFS man command examples

For more information about the `man` command, see
- `man — Display sections of the online reference manual` in z/OS UNIX System Services Command Reference
- Enabling the man pages in z/OS UNIX System Services Planning

ioeagfmt

Purpose

ioeagfmt is a batch utility that formats a VSAM linear data set to become an HFS compatibility mode aggregate. The aggregate can be either a version 1.4 or version 1.5 aggregate.
Format

```
ioeagfmt -aggregate name [-initialempty blocks] [-size blocks] [-logsize blocks] [-overwrite] [-compat]
[[-owner {uid|name}][-group {gid|name}]] [-perms {number}] [-grow blocks] [{-newauditfid|-nonewauditfid}]
[{-version4|-version5}] [-level] [-help]
```

Options

- **aggregate name**
  Specifies the name of the data set to format. This is also the aggregate name. The aggregate name is always translated to uppercase and cannot be longer than 44 characters. The following characters can be included in the name of an aggregate:
  - All uppercase and lowercase alphabetic characters (a to z, A to Z)
  - All numerals (0 to 9)
  - The . (period)
  - The - (dash)
  - The _ (underscore)
  - The @ (at sign)
  - The # (number sign)
  - The $ (dollar).

- **compat**
  Indicates that a compatibility mode aggregate should be created. This means that in addition to formatting the VSAM linear data set as a zFS aggregate, a zFS file system is created with the same name as the aggregate and its free space is set to the size of the available blocks on the aggregate. Beginning with z/OS V2R1, only HFS compatibility mode aggregates can be created. This option is being allowed for compatibility with earlier versions and is not needed.

- **group gid | name**
  Specifies the group owner for the root directory of the file system. It can be specified as a z/OS group name or as a GID. The default is the GID of the issuer of `ioeagfmt`. If only `-owner name` is specified, the group is that owner's default group. If only `-owner uid` is specified, the group is the issuer's group.

- **grow blocks**
  Specifies the number of 8-KB blocks that zFS will use as the increment for extension when the `-size` option specifies a size greater than the primary allocation.

- **help**
  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- **initialempty blocks**
  This option is being allowed for compatibility with earlier versions and is ignored. One 8-KB block at the beginning of the aggregate is reserved for IBM use.

- **level**
  Prints the level of the `ioeagfmt` command. This is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

- **logsize blocks**
  Specifies the size in 8-KB blocks of the log. The valid range is from 13 to 16384 blocks (128 megabytes). The default is 1% of the aggregate size. This default logsize will never be smaller than 14 blocks and it will never be
larger than 4096 blocks (32 megabytes). This size is normally sufficient. However, a small aggregate that is grown to be very large will still have a small log. You might want to specify a larger log if you expect the aggregate to grow very large.

- **newauditfid**
  Specifies that the aggregate should be formatted with the zFS auditfid and stored in the aggregate. Beginning with z/OS V2R1, -newauditfid is the default.

- **nnewauditfid**
  Specifies that the aggregate should not be formatted with a zFS auditfid stored in it. Before z/OS V2R1, this was the default.

- **overwrite**
  Required if you are reformatting an existing aggregate. Use this option with caution; it destroys any existing data. This option is not usually specified.

- **owner uid \userid**
  Specifies the owner for the root directory of the file system. It can be specified as a z/OS user ID or as a UID. The default is the UID of the issuer of `ioeagfmt`.

- **perms number**
  Specifies the permissions for the root directory of the file system. The number can be specified as octal (for example, o755), as hexadecimal (for example, x1ED), or as decimal (for example, 493). The default is o755 (owner read/write-execute, group read-execute, other read-execute).

- **size blocks**
  Specifies the number of 8-KB blocks that should be formatted to form the zFS aggregate. The default is the number of blocks that will fit in the primary allocation of the VSAM linear data set. If a number less than the default is specified, it is rounded up to the default. If a number greater than the default is specified, a single extend of the VSAM linear data set is attempted after the primary allocation is formatted unless the -grow option is specified. In that case, multiple extensions of the amount that is specified in the -grow option will be attempted until the -size is satisfied. The size can be rounded up to a control area (CA) boundary by DFSMS. It is not necessary to specify a secondary allocation size on the DEFINE of the VSAM linear data set for this extension to occur. Space must be available on the volume.

- **version4**
  Specifies that the aggregate should be a version 1.4 aggregate. See the Usage section for the default value that is used.

- **version5**
  Specifies that the aggregate should be a version 1.5 aggregate. See the Usage section for the default value that is used.

**Usage**

The `ioeagfmt` utility is used to format an existing VSAM linear data set as a zFS aggregate. All zFS aggregates must be formatted before use.

Beginning in z/OS V2R1, `ioeagfmt` fails if the zFS PFS is not active on the system.
ioeagfmt

The aggregate version will be as specified if the -version4 or -version5 parameter is used. If neither is used, then the default aggregate version will be obtained from the zFS PFS format_aggrversion setting. See "IOEFSPRM" on page 193 for a description of the format_aggrversion variable.

The size of the aggregate is as many 8-KB blocks as fits in the primary allocation of the VSAM linear data set or as specified in the -size option. The -size option can cause one additional extension to occur during formatting. To extend it further, use the zfsadm grow command. If -overwrite is specified, all existing primary and secondary allocations are formatted and the size includes all of that space. If the VSAM linear data set has a SHAREOPTIONS value of other than 3, ioeagfmt changes it to SHAREOPTIONS 3 during format. -overwrite will also cause the backup change activity flag to be set.

For a batch job, the ioeagfmt options are specified in the EXEC PARM as a single subparameter (a single character string enclosed in apostrophes with no commas separating the options). You cannot put the ending apostrophe in column 72. If it needs to go to the next line, use a continuation character in column 72 (continuing in column 16 with the ending apostrophe on the second line). Remember that a JCL EXEC PARM is limited to 100 characters. See the topic on the EXEC PARM in z/OS MVS JCL Reference.

Privilege required

If you are using an IOEFSPRM file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

The user must meet one of the following authorization requirements:

- Have ALTER authority to the VSAM linear data set
- Be UID 0
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

UPDATE authority to the VSAM linear data set is sufficient for format, but zFS will not be able to set the zFS bit in the catalog unless the issuer has ALTER authority.

If you are changing the owner or group to something other than the issuer or you are changing the permissions to other than the default, you need UID 0 or READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Examples

Figure 45 on page 129 shows an example of a job that creates a compatibility mode aggregate and file system.
ioeagslv

Purpose

ioeagslv is a batch utility that scans an aggregate and reports inconsistencies. Aggregates can be verified, recovered (that is, the log is replayed), or salvaged (that is, the aggregate is repaired). This utility is known as the salvager.

This utility is not normally needed. If a system failure occurs, the aggregate log is replayed automatically, the next time the aggregate is attached or mounted. This action typically brings the aggregate back to a consistent state. The aggregate must not be mounted (or attached) when ioeagslv is run.

Format

```
ioeagslv -aggregate name [{-recoveronly | -verifyonly | -salvageonly}] [-verbose] [-level] [-help]
```

Options

- **aggregate name**
  Specifies the name of the aggregate to be verified, recovered, or salvaged. The aggregate name is not case-sensitive. It is translated to uppercase.

- **help**
  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- **level**
  Prints the level of the ioeagslv command. This option is useful when you are diagnosing a problem. Except for -help, all other valid options that are specified with -level are ignored.

- **recoveronly**
  Directs the salvager to recover the specified aggregate. The salvager

Note: In the PARM=('-aggregate OMVS.PRV.COMPAT.AGGR001') statement, the -aggregate option must be in lowercase.
ioeagslv

replays the log of metadata changes that resides on the aggregate. See “Usage” on page 131 for information about using and combining the command's options.

-salvageonly
Directs the salvager to salvage the specified aggregate. The salvager attempts to repair any inconsistencies it finds on the aggregate. See “Usage” on page 131 for information about using and combining the command's options.

-verbose
This option is ignored.

-verifyonly
Directs the salvager to verify the specified aggregate. The salvager examines the structure of the aggregate to determine if it contains any inconsistencies, reporting any that it finds. See “Usage” on page 131 for information about using and combining the command's options.

Results

The salvager returns the following return codes for -verifyonly:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Success. The aggregate is correct and no repair is needed.</td>
</tr>
<tr>
<td>04</td>
<td>The aggregate has some inconsistencies that need repair.</td>
</tr>
<tr>
<td>08</td>
<td>An error occurred during verification; the report might be incomplete.</td>
</tr>
<tr>
<td>12</td>
<td>A severe error occurred during verification. Verify that processing was halted. The aggregate is not repairable.</td>
</tr>
<tr>
<td>16</td>
<td>Terminating error.</td>
</tr>
<tr>
<td>EIO</td>
<td>The salvager could not read or write the DASD.</td>
</tr>
<tr>
<td>EBUSY</td>
<td>The aggregate was mounted or attached.</td>
</tr>
<tr>
<td>EMVSERR</td>
<td>The salvager had an internal error. This return code is preceded by a dump for an abend 2C3 and reason code EA660701.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>The salvager ran out of storage.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>The salvager arguments were incorrect.</td>
</tr>
<tr>
<td>ENOSPC</td>
<td>Dynamic grow failed because the salvager ran out of disk space.</td>
</tr>
</tbody>
</table>

For no options specified (or the -recoveronly and -salvageonly options specified) the salvager returns the following return codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Success. The aggregate is correct and no repair is needed.</td>
</tr>
<tr>
<td>04</td>
<td>The aggregate had some inconsistencies that were repaired.</td>
</tr>
<tr>
<td>08</td>
<td>An error occurred during verification; the report might be incomplete; the aggregate could not be repaired.</td>
</tr>
<tr>
<td>12</td>
<td>A severe error occurred during verification and the aggregate could not be repaired. Verification processing was stopped.</td>
</tr>
</tbody>
</table>
16 Terminating error.
EIO The salvager could not read or write the DASD.
EBUSY The aggregate was mounted or attached.
EMVSERR The salvager had an internal error. This return code is preceded by a dump for an abend 2C3 and reason code EA660701.
ENOMEM The salvager ran out of storage.
EINVAL The salvager arguments were incorrect.

Usage

You can run `ioeagslv` even if the zFS PFS is not active on the system. The `ioeagslv` utility invokes the salvager on the zFS aggregate that is specified with the `-aggregate` option. After a system restart, the salvager employs the zFS file system log mechanism to return consistency to a file system by running recovery on the aggregate on which the file system resides. Recovery is the replaying of the log on the aggregate; the log records all changes that are made to metadata as a result of operations such as file creation and deletion. If problems are detected in the basic structure of the aggregate, if the log mechanism is damaged, or if the storage medium of the aggregate is suspect, the `ioeagslv` utility must be used to verify or repair the structure of the aggregate.

Use the utility's `-recoveryonly`, `-verifyonly`, and `-salvageonly` options to indicate the operations the salvager is to perform on the specified aggregate, as follows:

- Specify the `-recoveryonly` option
  To run recovery on the aggregate without attempting to find or repair any inconsistencies found on it. Recovery is the replaying of the log on the aggregate. Use this option to quickly return consistency to an aggregate that does not need to be salvaged; this represents the normal production use of the salvager. Unless the contents of the log or the physical structure of the aggregate is damaged, replaying the log is an effective guarantee of a file system's integrity.

- Specify the `-verifyonly` option
  To determine whether the structure of the aggregate contains any inconsistencies. Use this option to assess the extent of the damage to an aggregate. The salvager runs log recovery and then determines whether there are any inconsistencies. No repair is attempted other than running log recovery.

- Specify the `-salvageonly` option
  To attempt to repair any inconsistencies that are found in the structure of the aggregate without first running recovery on it. Use this option if you believe the log is damaged or replaying the log does not return consistency to the aggregate and might in fact further damage it. In most cases, you do not salvage an aggregate without first recovering it.

- Omit the `-recoveryonly`, `-verifyonly`, and `-salvageonly` options
  To run recovery on the aggregate and then attempt to repair any inconsistencies that are found in the structure of the aggregate. Because recovery eliminates inconsistencies in an undamaged file system, an aggregate is typically recovered.
before it is salvaged. In general, it is good first to recover and then to salvage an aggregate if a system goes down or experiences a hardware failure.

Omit these three options if you believe the log should be replayed before attempts are made to repair any inconsistencies that are found on the aggregate. (Omitting the three options is equivalent to specifying the -recoveronly and -salvageonly options.)

The salvager utility can set or clear the aggregate damaged bit:

- The -verifyonly option can set the bit if a true corruption is found or clear it if no corruption is found.
- Repair (with no option) can clear the bit if a successful repair is done.

The following rule summarizes the interaction of the -recoveronly, -verifyonly, and -salvageonly options: The salvage command runs recovery on an aggregate and attempts to repair it unless one of the three salvage options is specified; after one of these options is specified, you must explicitly request any operation that you want the salvager to perform on the aggregate.

The basic function of the salvager is similar to that of the fsck program in many UNIX systems. The salvager recovers a zFS aggregate and repairs problems it detects in the structure of the aggregate. It does not verify or repair the format of user data that is contained in files on the aggregate.

The salvager verifies the structure of an aggregate by examining all of the anodes, directories, and other metadata in each file system on the aggregate. An anode is an area on the disk that provides information that is used to locate data such as files, directories, ACLs, and other types of file system objects. Each file system contains an arbitrary number of anodes, all of which must reside on the same aggregate. By following the links between the various types of anodes, the salvager can determine whether the organization of an aggregate and the file system it contains is correct and make repairs if necessary.

The salvager is designed to make all repairs in one pass, but due to the nature of the program's inputs (a corrupted, possibly vastly corrupted file system) IBM recommends a second running of the salvage program to verify that the aggregate is truly repaired. If verifying the aggregate shows that it is not repaired, then you should try running the salvager again to repair the aggregate. If this does not repair the aggregate, you can create a copy of the aggregate and run the salvager more times to try and repair it. If the salvager cannot repair the aggregate after several repair attempts, the copy of the aggregate and salvager job logs will allow IBM service to determine why.

Not all aggregates can be salvaged. In cases of extensive damage to the structure of the metadata on an aggregate or damage to the physical disk that houses an aggregate, the salvager cannot repair inconsistencies. Also, the salvager cannot verify or repair damage to user data on an aggregate. The salvager cannot detect problems that modified the contents of a file but did not damage the structure of an aggregate or change the metadata of the aggregate.

Like the fsck command, the salvager analyzes the consistency of an aggregate by making successive passes through the aggregate. With each successive pass, the salvager examines and extracts a different type of information from the blocks and anodes on the aggregate. Later passes of the salvager use information that is found in earlier passes to help in the analysis.
It is possible for the salvager to attempt a dynamic grow of an aggregate. One possible reason for this is if an extended (v5) directory is found to be inconsistent (or broken). The salvager will try to repair it by converting it to a new extended (v5) directory. To do this might require more disk space. If the disk space is not available, the directory is marked read-only. The rest of the file system has already been made consistent, so you should still be able to mount the file system and read from the directory.

In general, if the salvager is invoked for a VSAM linear data set that it is sure is not a zFS aggregate, it exits with an error code of at least 16 without analyzing the VSAM linear data set. It exits with an error code of EBUSY (114) if a file system on the aggregate to be recovered or salvaged is mounted or attached. (If necessary, you can use the UNMOUNT command to unmount the aggregate.)

Beginning in z/OS V2R1, the salvager no longer supports salvaging aggregates that contain more than one file system or clones (.bak file systems). For additional details about running the salvage utility, see "Understanding the salvager utility" on page 99.

As the salvager runs, it maintains a list of sorted error records that need repair. Each record includes details for the salvager to quickly repair the aggregate. The salvager displays corruption messages if verification found any inconsistency. It also displays progress messages (IOEZ00782I) during verification to indicate how many objects have been processed. Depending on the aggregate size and system usage, the salvager batch job might take hours or even longer to complete.

For a batch job, the `ioeagslv` options are specified in the EXEC PARM as a single subparameter (a single character string enclosed in apostrophes with no commas separating the options). You cannot put the ending apostrophe in column 72. If it needs go to the next line, use a continuation character in column 72 (continuing in column 16 with the ending apostrophe on the second line). Remember that a JCL EXEC PARM is limited to 100 characters. See the topic on the EXEC PARM in z/OS MVS JCL Reference. For an example of the EXEC PARM for `ioeagslv`, see Figure 46 on page 134.

The zFS configuration file can include debugging parameters for the salvager utility. The debugging parameters are described in "IOEFSprm" on page 193. There are two ways that you can implement the configuration file:

- As a single file that is defined by a IOEZPRM DD card
- As one or more parameter file members, named IOEPRMxx

You can provide an optional IOEZPRM DD statement in the JCL for the batch job to specify the location of the IOEFSprm file. Or, you can omit the IOEZPRM DD statement and specify the -PRM option on the EXEC PARM to use IOEPRMxx parameter file members. If you do not specify the IOEZPRM DD statement, the utility searches the logical parmlib concatenation to find the IOEPRMxx members that contain the debugging parameters, in the same way that the zFS PFS does if you do not specify the IOEZPRM DD statement in the zFS PROC. For more information about specifying the configuration file, see "IOEFSprm" on page 193.

`ioeagslv` causes the backup change activity flag to be set if the log is replayed or a repair is done.

`ioeagslv` can be used to salvage aggregate versions 1.4 and 1.5.
Privilege required

The user must meet one of the authorization requirements:

• Have UPDATE authority for the specified VSAM linear data set.
• Be UID 0.
• Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

If you are using an IOEFSPRM file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxxx), the issuer does not need special authorization.

Examples

Figure 46 shows an example of a job that invokes the ioeagslv utility.

```plaintext
//USERID A JOB , 'Salvage',
 // CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//SALVAGE EXEC PGM=IOEAGSLV,REGION=0M,
// PARM=(' -aggregate OMVS.PRIV.COMPAT.AGGR001 -verifyonly')
//IOEZPRM DD DSN=SYS4.PVT.SY1.PARMLIB(IOEFSPRM),DISP=SHR
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
```  

Figure 46. Job to verify a zFS aggregate using debug parameters specified in IOEFSPRM

```plaintext
//USERIDA JOB ,'Salvage',
 // CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//SALVAGE EXEC PGM=IOEAGSLV,REGION=0M,
// PARM=(' -aggregate OMVS.PRIV.COMPAT.AGGR001 -verifyonly -PRM=(03)')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
```  

Figure 47. Job to verify a zFS aggregate using debug parameters specified in parmlib member IOEPRM03

ioefsutl

Purpose

This section introduces the ioefsutl batch utility suite. It is run as a batch job. A zFS aggregate must be unmounted (and not attached) before ioefsutl can process it.

ioefsutl is a batch utility that supports the following functions:

• format of a new aggregate in the specified version.
• salvage to verify and repair a damaged aggregate.
ioefsutl

- `converttov5` to change a version 1.4 aggregate to a version 1.5 aggregate and convert all the existing directories to extended (v5) directories.
- `converttov4` to convert all extended (v5) directories to v4 directories and then change the version 1.5 aggregate to a version 1.4 aggregate.

If you are using the IOEFSPRM file, you can provide an optional IOEZPRM DD statement in the JCL for a batch job to specify the location of the IOEFSPRM file. If you are using the IOEPRMxx parmlib member, omit the IOEZPRM DD statement and specify the `-PRM` option on the EXEC PARM; for example, `-PRM=(03)` if your configuration file is in the parmlib member IOEPRM03. If you do not specify the IOEZPRM DD statement, the utility searches the logical parmlib concatenation to find the IOEPRMxx members that contain the debugging parameters, in the same way that the zFS PFS does if you do not specify the IOEZPRM DD statement in the ZFS PROC. For more information about specifying the configuration file, see “IOEFSPRM” on page 193.

---

### ioefsutl converttov4

#### Purpose

`ioefsutl converttov4` is a batch utility that converts a version 1.5 aggregate to a version 1.4 aggregate.

#### Format

```
ioefsutl converttov4 -aggregate name [-verbose][-level][-help]
```

#### Options

- `-aggregate` *name*
  
  Specifies the name of the aggregate to be converted. The aggregate name is not case-sensitive. It is translated to uppercase.

- `-help`
  
  Prints the online help for this command. All other valid options specified with this option are ignored.

- `-level`
  
  Prints the level of the `ioefsutl` command. This information is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

- `-verbose`
  
  Displays starting and ending messages of each directory being converted.

#### Usage

The `ioefsutl converttov4` command is used when you need to convert a zFS version 1.5 aggregate to a version 1.4 aggregate. All extended (v5) directories are converted to v4 directories. You might use this if you need to run z/OS releases prior to z/OS V2R1.

`ioefsutl converttov4` cannot convert the version 1.5 aggregate if it has grown larger than approximately 4 TB. In this case, you must copy subsets of the data one at a time into other version 1.4 aggregates using the z/OS UNIX shell command `pax`. Each subset must be copied into a separate version 1.4 aggregate that is less than 4 TB.
ioefsutl converttov4

ioefsutl converttov4 cannot convert a directory that contains more than 64K-1 subdirectories. In this case, you must copy subsets of the directory into separate directories contained in a version 1.4 aggregate.

Converting a directory from an extended (v5) directory to a version 4 directory requires both versions of the directory to be on the disk at the same time, temporarily. If the aggregate becomes full during the allocation of the new directory, a dynamic grow is attempted. See "Dynamically growing a compatibility mode aggregate" on page 28 for information about controlling the dynamic growth of an aggregate. If there is not enough space to complete the conversion, the new directory is deleted and the conversion operation fails.

The old directory is deleted when the conversion is completed. The resulting new directory can possibly require more space than the old directory, and could also possibly require less space than the old directory. Results will vary based on the actual directory contents.

If a system outage occurs during a directory conversion, the directory is made consistent during log recovery processing. That is, either the old directory will exist or the new directory will exist, but both will not exist.

The conversion will cause the backup change activity flag to be set.

If the aggregate damaged bit is set, conversion does not start and an error is issued.

If the aggregate damaged bit is set, you can still mount the aggregate. The IOEZ00783E console message is displayed:

IOEZ00783E Aggregate aggregate_name is damaged

Privilege required

The user must meet one of the authorization requirements:

- Have UPDATE authority for the specified VSAM linear data set
- Be UID 0
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class

If you are using an IOEFSPRM file in your JCL, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

Figure 48 on page 137 shows an example of a job that invokes the ioefsutl utility to convert a version 1.5 aggregate to a version 1.4 aggregate.
In the PARM=('converttov4 -aggregate OMVS.PRV.COMPAT.AGGR001') statement, the `converttov4` and option `-aggregate` must be in lowercase.

---

**ioefsutl converttov5**

**Purpose**

`ioefsutl converttov5` is a batch utility that converts a version 1.4 aggregate to a version 1.5 aggregate.

**Format**

```
ioefsutl converttov5 -aggregate name -aggrversion_only [-verbose][-level][-help]
```

**Options**

- **-aggregate name**
  
  Specifies the name of the aggregate to be converted. The aggregate name is not case-sensitive. It is translated to uppercase.

- **-aggrversion_only**
  
  Only the aggregate version is converted from 1.4 to 1.5. No directories are converted.

- **-help**
  
  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- **-level**
  
  Prints the level of the `ioefsutl` command. This information is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

- **-verbose**
  
  Displays starting and ending messages of each directory being converted.

**Usage**

The `ioefsutl converttov5` command is used when you need to convert a zFS version 1.4 aggregate to a version 1.5 aggregate. All v4 directories are converted to extended (v5) directories. You might use this if you have migrated all your systems to z/OS V2R1 and you want to exploit extended (v5) directories.

Converting a directory from version 4 to an extended (v5) directory requires both versions of the directory to exist on disk at the same time, temporarily. If the aggregate becomes full during the allocation of the new directory a dynamic grow...
will be attempted. See “Dynamically growing a compatibility mode aggregate” on page 28 for information about controlling dynamic growth of an aggregate. If there is not enough space to complete the conversion the new directory will be deleted and the conversion operation will fail.

When the conversion is completed the old directory will be deleted. The resulting new directory can possibly require more space than the old directory, and could also possibly require less space than the old directory. Results will vary based on the actual directory contents.

If a system outage occurs during a directory conversion, the directory will be made consistent during log recovery processing. That is, either the old directory will exist or the new directory will exist, but both will not exist.

The conversion will cause the backup change activity flag to be set.

If the aggregate damaged bit is set, conversion does not start and an error is issued.

If the aggregate damaged bit is set, you can still mount the aggregate. The IOEZ00783E console message is displayed:

IOEZ00783E Aggregate aggregate_name is damaged

Privilege required

The user must meet one of the authorization requirements:

- Have UPDATE authority for the specified VSAM linear data set.
- Be UID 0.
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

If you are using an IOEFSPRM file in your JCL, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

Figure 49 shows an example of a job that invokes the ioefsutl utility to convert a version 1.4 aggregate to a version 1.5 aggregate.

```
//USERIDA JOB ,,'Convert to version 5',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//CONVERT EXEC PGM=IOEFSUTL,REGION=0M,
// PARM=('converttov5 -aggregate OMVS.PRIV.COMPAT.AGGR001')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*
```

Figure 49. Job to convert a version 1.4 aggregate to a version 1.5 aggregate

In the PARM=('converttov5 -aggregate OMVS.PRIV.COMPAT.AGGR001') statement, the converttov5 and option -aggregate must be in lowercase.
ioefsutl format

Purpose

ioefsutl format is a batch utility that formats a VSAM linear data set to become a zFS compatibility mode aggregate.

Format

```
ioefsutl format -aggregate name [-size blocks] [-logsize blocks] [-owner uid|name]
[-group gid|name] [-perms number] [-grow blocks][-overwrite][[-version4 | -version5]]
[-level][-help]
```

Options

-aggregate name
  Specifies the name of the data set to format. This is also the aggregate name. The aggregate name is always translated to uppercase and cannot be longer than 44 characters. The following characters can be included in the name of an aggregate:
  • All uppercase and lowercase alphabetic characters (a to z, A to Z)
  • All numerals (0 to 9)
  • The . (period)
  • The - (dash)
  • The _ (underscore)
  • The @ (at sign)
  • The # (number sign)
  • The $ (dollar).

-group gid | name
  Specifies the group owner for the root directory of the file system. It can be specified as a z/OS group name or as a GID. The default is the GID of the issuer of ioefsutl format. If only -owner name is specified, the group is that owner's default group. If only -owner uid is specified, the group is the issuer's group.

-grow blocks
  Specifies the number of 8-KB blocks that zFS uses as the increment for extension when the -size option specifies a size greater than the primary allocation.

-help
  Prints the online help for this command. All other valid options that are specified with this option are ignored.

-logsize blocks
  Specifies the size in 8-KB blocks of the log. The valid range is from 13 to 16384 blocks (128 megabytes). The default is 1% of the aggregate size. This default logsize will never be smaller than 14 blocks and it will never be larger than 4096 blocks (32 megabytes). This size is normally sufficient. However, a small aggregate that is grown to be very large will still have a small log. You might want to specify a larger log if you expect the aggregate to grow very large.

-level
  Prints the level of the ioefsutl command. This information is useful when you are diagnosing a problem. Except for -help, all other valid options that are specified with -level are ignored.
ioefsutl format

-overwrite
  Required if you are reformatting an existing aggregate. Use this option with caution because it destroys any existing data. This option is not usually specified.

-owner uid 1 name
  Specifies the owner for the root directory of the file system. It can be specified as a z/OS user ID or as a UID. The default is the UID of the issuer of ioefsutl format.

-perms number
  Specifies the permissions for the root directory of the file system. The number can be specified as octal (for example, o755), as hexadecimal (for example, x1ED), or as decimal (for example, 493). The default is o755 (owner read/write-execute, group read-execute, other read-execute).

-size blocks
  Specifies the number of 8-KB blocks that should be formatted to form the zFS aggregate. The default is the number of blocks that will fit in the primary allocation of the VSAM linear data set. If a number less than the default is specified, it is rounded up to the default. If a number greater than the default is specified, a single extend of the VSAM linear data set is attempted after the primary allocation is formatted unless the -grow option is specified. In that case, multiple extensions of the amount that is specified in the -grow option will be attempted until the -size is satisfied. The size can be rounded up to a control area (CA) boundary by DFSMS. It is not necessary to specify a secondary allocation size on the DEFINE of the VSAM linear data set for this extension to occur. Space must be available on the volume.

-version4
  Specifies that the aggregate is to be formatted as a version 1.4 aggregate. See the Usage section for information about how the default aggregate version is determined.

-version5
  Specifies that the aggregate is to be formatted as a version 1.5 aggregate. See the Usage section for information about how the default aggregate version is determined. IBM recommends that you do not use -version5 until all your systems are at z/OS V2R1.

Usage

The aggregate name is not case-sensitive. It is translated to uppercase. If -version4 or -version5 is specified, you can run ioefsutl format even if the zFS PFS is not active on the system. If neither is specified, the aggregate version default is determined by a call to the zFS PFS to obtain the value of the format_aggrversion variable from the IOEFSPRM file. If the zFS PFS is not active, then the format will fail.

The size of the aggregate is as many 8K blocks as fits in the primary allocation of the VSAM linear data set or as specified in the -size option. The -size option can cause one additional extension to occur during formatting. To extend it further, use the zfsadm grow command. If -overwrite is specified, all existing primary and secondary allocations are formatted and the size includes all of that space. If -overwrite is specified, the backup change activity flag will be set. If the VSAM linear data set has a SHAREOPTIONS value of other than 3, ioefsutl format changes it to SHAREOPTIONS 3 during format.
For a batch job, the `ioefsutl format` options are specified in the EXEC PARM as a single subparameter (a single character string enclosed in apostrophes with no commas separating the options). You cannot put the ending apostrophe in column 72. If it needs to go to the next line, use a continuation character in column 72 (continuing in column 16 with the ending apostrophe on the second line). Remember that a JCL EXEC PARM is limited to 100 characters. See the topic on EXEC PARM in z/OS MVS JCL Reference.

`ioefsutl format` always formats with a unique auditfid.

**Privilege required**

The user must meet one of the following authorization requirements:

- Have ALTER authority to the VSAM linear data set.
- Be UID 0.
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

UPDATE authority to the VSAM linear data set is sufficient for format, but zFS will not be able to set the zFS bit in the catalog unless the issuer has ALTER authority.

If you are changing the owner or group to something other than the issuer or you are changing the permissions to other than the default, you need UID 0 or READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

If you are using an IOEFSprm file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSprm file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

**Restrictions**

The zFS aggregate cannot be mounted (or attached). The batch job must be issued from a V2R1 system and the VSAM linear data set must exist. If neither `-version4` nor `-version5` are specified, the value of the `format_aggrversion` parameter on the server is used. In this case, if the value of the `FORMAT_AGGRVERSION` parameter cannot be determined, the format will fail.

**Examples**

Figure 50 on page 142 shows an example of a job that creates and formats a version 1.4 aggregate.
Note: In the PARM=('format -aggregate OMVS.PRV.COMPAT.AGGR001 -version4') statement, the format, and options -aggregate and -version4 must be in lowercase.

ioefsutl salvage

**Purpose**

ioefsutl salvage is a batch utility that scans an aggregate and reports inconsistencies. Aggregates can be verified, recovered (that is, the log is replayed), or salvaged (that is, the aggregate is repaired). This utility is known as the salvager.

This utility is not normally needed. If a system failure occurs, the aggregate log is replayed automatically the next time the aggregate is attached or mounted. This action typically brings the aggregate back to a consistent state. The aggregate must not be mounted (or attached) when ioefsutl salvage is run.

**Format**

```
ioefsutl salvage -aggregate name [-verifyonly][][-level][[-help]
```

**Options**

- **-aggregate name**
  Specifies the name of the aggregate to be verified or salvaged. The aggregate name is not case-sensitive. It is translated to uppercase.

- **-help**
  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- **-verifyonly**
  Specifies that the salvager is to verify the specified aggregate. It should not attempt to repair any damage found. The log is replayed before the verification unless an error occurs during the replay. If this option is
omitted, the salvager will replay the log, verify the specified aggregate and
then attempt to repair any damage found.

-`level` Prints the level of the `ioefsutl` command. This information is useful when
you are diagnosing a problem. Except for `-help`, all other valid options that
are specified with `-level` are ignored.

Results

For `-verifyonly`, the salvager returns the following return codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Success. The aggregate is correct and no repair is needed.</td>
</tr>
<tr>
<td>04</td>
<td>The aggregate has some inconsistencies that need repair.</td>
</tr>
<tr>
<td>08</td>
<td>An error occurred during verification; the report might be incomplete.</td>
</tr>
<tr>
<td>12</td>
<td>A severe error occurred during verification. Verify that processing was halted. The aggregate is not repairable.</td>
</tr>
<tr>
<td>16</td>
<td>Terminating error.</td>
</tr>
<tr>
<td>EIO</td>
<td>The salvager could not read or write the DASD.</td>
</tr>
<tr>
<td>EBUSY</td>
<td>The aggregate was mounted or attached.</td>
</tr>
<tr>
<td>EMVSERR</td>
<td>The salvager had an internal error. This return code is preceded by a dump for an abend 2C3 and reason code EA660701.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>The salvager ran out of storage.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>The salvager arguments were incorrect.</td>
</tr>
</tbody>
</table>

For no options specified, the salvager returns the following return codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Success. The aggregate is correct and no repair is needed.</td>
</tr>
<tr>
<td>04</td>
<td>The aggregate had some inconsistencies that were repaired.</td>
</tr>
<tr>
<td>08</td>
<td>An error occurred during verification; the report might be incomplete; the aggregate could not be repaired.</td>
</tr>
<tr>
<td>12</td>
<td>A severe error occurred during verification; verify processing halted; the aggregate could not be repaired.</td>
</tr>
<tr>
<td>16</td>
<td>Terminating error.</td>
</tr>
<tr>
<td>EIO</td>
<td>The salvager could not read or write the DASD.</td>
</tr>
<tr>
<td>EBUSY</td>
<td>The aggregate was mounted or attached.</td>
</tr>
<tr>
<td>EMVSERR</td>
<td>The salvager had an internal error. This return code is preceded by a dump for an abend 2C3 and reason code EA660701.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>The salvager ran out of storage.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>The salvager arguments were incorrect.</td>
</tr>
</tbody>
</table>
ioefsutl salvage

Usage

You can run `ioefsutl salvage` even if the zFS PFS is not active on the system. The `ioefsutl salvage` utility invokes the salvager on the zFS aggregate that is specified with the `-aggregate` option.

The salvager cannot process an aggregate that contains multiple file systems or a clone.

The processing of the aggregate is controlled by the specification or the omission of the `-verifyonly` option.

- Specify the `-verifyonly` option
  To determine whether the structure of the aggregate contains any inconsistencies. Use this option to assess the extent of the damage to an aggregate. The salvager runs log recovery and then determines whether there are any inconsistencies. No repair is attempted other than running log recovery.

- Omit the `-verifyonly` option
  To run log recovery on the aggregate, verify the aggregate and then attempt to repair any inconsistencies that are found in the structure of the aggregate. Because log recovery eliminates inconsistencies in an undamaged file system, an aggregate is typically recovered before it is salvaged. In general, it is good practice to first recover and then to salvage an aggregate if a system goes down or experiences a hardware failure.

The salvager sets the backup change activity flag if log recovery is run or a repair is done.

The basic function of the salvager is similar to that of the `fsck` program in many UNIX systems. The salvager recovers a zFS aggregate and repairs problems it detects in the structure of the aggregate. It does not verify or repair the format of user data that is contained in files on the aggregate.

The salvager verifies the structure of an aggregate by examining all of the anodes, directories, and other metadata in each file system on the aggregate. An anode is an area on the disk that provides information that is used to locate data such as files, directories, ACLs, and other types of file system objects. Each file system contains an arbitrary number of anodes, all of which must reside on the same aggregate. By following the links between the various types of anodes, the salvager can determine whether the organization of an aggregate and the file system that it contains is correct and make repairs if necessary.

Not all aggregates can be salvaged. In cases of extensive damage to the structure of the metadata on an aggregate or damage to the physical disk that houses an aggregate, the salvager cannot repair inconsistencies. Also, the salvager cannot verify or repair damage to user data on an aggregate. The salvager cannot detect problems that modified the contents of a file but did not damage the structure of an aggregate or change the metadata of the aggregate.

The salvager is designed to make all repairs in one pass, but due to the nature of the program's inputs (a corrupted, possibly vastly corrupted file system) IBM recommends a second running of the salvage program to verify that the aggregate is truly repaired. If verifying the aggregate shows that it is not repaired, then you should try running the salvager again to repair the aggregate. If this does not repair the aggregate, you can create a copy of the aggregate and run the salvager
more times to try and repair it. If the salvager cannot repair the aggregate after several repair attempts, the copy of the aggregate and salvager job logs will allow IBM service to determine why.

Like the fsck command, the salvager analyzes the consistency of an aggregate by making successive passes through the aggregate. With each successive pass, the salvager examines and extracts a different type of information from the blocks and anodes on the aggregate. Later passes of the salvager use information that was found in earlier passes to help in the analysis.

It is possible for the salvager to attempt a dynamic grow of an aggregate. One possible reason for this is if an extended (v5) directory is found to be inconsistent (or broken). The salvager will try to repair it by converting it to a new extended (v5) directory. To do this might require more disk space. If the disk space is not available the directory is marked read-only. The rest of the file system has already been made consistent, so you should still be able to mount the file system and read from the directory.

In general, if the salvager is invoked for a VSAM linear data set that it is sure is not a zFS aggregate, it exits with an error code of at least 16 without analyzing the VSAM linear data set. It exits with an error code of EBUSY (114) if a file system on the aggregate to be recovered or salvaged is mounted or attached. (If necessary, you can use the unmount command to unmount the aggregate.)

As the salvager runs, it maintains a list of sorted error records that need repair. Each record includes details for the salvager to quickly repair the aggregate. The salvager displays corruption messages if verification found any inconsistencies. It also displays progress messages (IOEZ00782I) during verification to indicate how many objects were processed. Depending on the aggregate size and system usage, the salvager batch job may take hours or even longer to complete.

For more information about running the salvage utility, see "Understanding the salvager utility" on page 99.

For a batch job, the ioefsutl salvage options are specified in the EXEC PARM as a single subparameter (a single character string enclosed in apostrophes with no commas separating the options). You cannot put the ending apostrophe in column 72. If it needs to go to the next line, use a continuation character in column 72 (continuing in column 16 with the ending apostrophe on the second line). Remember that a JCL EXEC PARM is limited to 100 characters. See the topic on the EXEC PARM in z/OS MVS JCL Reference. For an example of the EXEC PARM for ioefsutl salvage, see Figure 51 on page 146.

ioefsutl salvage can be used to salvage aggregate versions 1.4 and 1.5.

The salvager utility can set or clear the aggregate damaged bit:
• The -verifyonly option can set the bit if a true corruption is found or clear it if no corruption is found.
• Repair (with no option) can clear the bit if a successful repair is done.

Privilege required

The user must meet one of the following authorization requirements:
• Have UPDATE authority to the VSAM linear data set
• Be UID 0
ioefsutl salvage

- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class

If you are using an IOEFSPRM file in the JCL, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

Figure 51 shows an example of a job to salvage a zFS aggregate:

```
//USERIDA JOB ,'Salvage verify'.
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//SALVAGE EXEC PGM=IOEFSUTL,REGION=0M,
// PARM=('salvage -aggregate OMVS.PRIV.COMPAT.AGGR001 -verifyonly')
//IOEZPRM DD DSN=SYS4.PVT.SY1.PARMLIB(IOEFSPRM),DISP=SHR
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*
```

Figure 51. Job to verify a zFS aggregate using debug parameters specified in IOEZPRM

Note: In the PARM=('salvage -aggregate OMVS.PRIV.COMPAT.AGGR001 -verifyonly') statement, the salvage and options -aggregate and -verifyonly must be in lowercase.

MOUNT

Purpose

MOUNT is a TSO/E command that mounts a file system into the z/OS UNIX hierarchy. This section only documents MOUNT options that are unique to zFS. It can also be invoked from the z/OS UNIX shell (/usr/sbin/mount). For additional information about this command, see z/OS UNIX System Services Command Reference.

Note:
1. Beginning with z/OS V2R1, zFS clones are no longer supported. An attempt to mount an aggregate that contains a .bak (clone) file system will be denied.
2. Beginning with z/OS V2R1, multi-file system aggregates are no longer supported. An attempt to mount a zFS file system that is contained in a zFS multi-file system aggregate is denied.

Format

```
MOUNT TYPE(file_system_type) [PARM(parameter_string)]
```

Options

**TYPE (file_system_type)**
Specifies the file system type. Specify ZFS or HFS and the correct file system type is determined for the file system that is located by the data set
name. If the TYPE specified (HFS) does not match the real file system type (ZFS), any associated ZFS parameters are ignored. For more information, see [Mounting considerations](z/OS UNIX System Services Planning).

**PARM**(parameter_string)

Specifies a parameter string to be passed to zFS. Parameters are case-sensitive and separated by a comma. Enclose the parameter string within quotation marks. If a parameter is specified multiple times, the last parameter is used.

**Note:** If the value specified on the TYPE parameter (HFS) does not match the real file system type (ZFS), any associated ZFS parameters are ignored.

**AGGRFULL**(threshold,increment)

Specifies the threshold and increment for reporting aggregate full error messages to the operator. The default is the aggrfull specification in the IOEFSprm file.

**Note:** AGGRFULL and FSFULL provide the same function. You can use either one (or both) to monitor the free space for an aggregate. However, AGGRFULL tends to give a more accurate view of free space and is the suggested choice.

**AGGRGROW | NOAGGRGROW**

Specifies whether the aggregate is eligible to be dynamically grown. The growth is based on the secondary allocation of the aggregate and will occur when the aggregate becomes full. The default is the aggrgrow specification in the IOEFSprm file.

**CONVERTTOV5 | NOCONVERTTOV5**

Specifies whether a zFS read/write file system is assigned the converttov5 attribute. If it is assigned the converttov5 attribute and the aggregate is a version 1.5 aggregate, zFS automatically converts directories from v4 to extended (v5) as they are accessed. If the converttov5 attribute is assigned at primary mount time, a version 1.4 aggregate is changed to a version 1.5 aggregate.

If automatic directory conversion for a directory fails, the conversion is not attempted again until the file system is unmounted and mounted again.

The converttov5 attribute can also be assigned if the MOUNT option is not specified but the converttov5 specification in the IOEFSprm file is on when the file system is mounted or remounted.

The default is NOCONVERTTOV5. However, the converttov5 attribute can also be assigned if the converttov5 specification in the IOEFSprm file is on when the file system is mounted or remounted.

**FSFULL**(threshold,increment)

Specifies the threshold and increment for reporting file system free space error messages to the operator. The default is the fsfull specification in the IOEFSprm file.

**Note:** AGGRFULL and FSFULL provide the same function. You can use either one (or both) to monitor the free space for an aggregate. However, AGGRFULL tends to give a more accurate view of free space and is the suggested choice.
RWSHARE | NORWSHARE
Specifies whether a zFS read/write mounted file system will be mounted sysplex-aware or non-sysplex aware. zFS must be running sysplex-aware on a file system basis (IOEFSPRM specifies sysplex=filesys) for this parameter to take effect. The default is the sysplex_filesys_sharemode specified in the IOEFSPRM file, or later using the zfsadm config command. For information about whether to make a read/write file system sysplex aware, see "Using zFS read/write sysplex-aware file systems" on page 15.

Usage
A mount of a compatibility mode aggregate is serialized with other zfsadm commands (because the mount of a compatibility mode aggregate does an implicit attach).

If you attempt to mount a compatibility mode aggregate/file system read-only and it fails because it needs to run recovery (return code EROFS (141) and reason code EFxx6271), you should temporarily mount it read/write (so it can complete the recovery process) and then mount it read-only. Alternatively, you can specify the romount_recovery=on configuration option in IOEFSPRM. This causes the file system to automatically be temporarily mounted read/write to allow log recovery to run and then to be mounted read-only.

If the DASD volume containing the zFS compatibility mode aggregate being mounted is read-only, you can receive message IOEZ00336I. This message indicates that the zFS aggregate indicator cannot be set in the catalog (actually, in the VVDS on the volume). The zFS aggregate is successfully mounted (and attached). DFSMSdss backup (DUMP) will not automatically quiesce and unquiesce the zFS aggregate because it cannot determine that the VSAM linear data set is a zFS aggregate. If the zFS aggregate can be mounted with the DASD volume in read/write, the zFS aggregate indicator will be set.

You can determine if the zFS aggregate indicator is set by using IDCAMS LISTCAT ALL against the zFS aggregate and looking for the zFS indicator in the output.

Do not use a path entry as the file system name in the MOUNT command (see the topic on DEFINE PATH in z/OS DFSMS Access Method Services Commands). The mount succeeds but the system issues messages similar to the following:

IOEZ00412I Catalog search failed for aggregate PLEX.JMS.AGGR006.PATH. Shareoptions are not altered.
IOEZ00336I PLEX.JMS.AGGR006.PATH could not be marked as a zFS aggregate in the catalog, rc=60 rsn=104

Examples
The following TSO/E example mounts a zFS file system and specifies a threshold and increment to display a message when the file system becomes almost full:

MOUNT FILESYSTEM('OMVS.PRV.AGGR004.LDS0004') MOUNTPOINT('/etc/zfscompat1')
TYPE(ZFS) MODE(RDWR) PARM('AGGRFULL(90,5)')
zfsadm

Purpose

This section introduces the zfsadm command suite. The zfsadm command is run from the z/OS UNIX shell. It can also be invoked from TSO/E by using the program name IOEZADM or as a batch job by using PGM=IOEZADM. See Figure 53 on page 158 for an example of invoking IOEZADM from a batch job.

Command syntax

The zfsadm commands have the same general structure:

```
command {-option1 argument... | -option2 {argument1 | argument2}...} [-optional_information]
```

The following example illustrates the elements of a zfsadm command:

```
zfsadm detach {-all | -aggregate name} [-help]
```

The following list summarizes the elements of the zfsadm command:

Command

A command consists of the command suite (zfsadm in the previous example) and the command name (detach). The command suite and the command name must be separated by a space. The command suite specifies the group of related commands.

Options

Command options always appear in monospace type in the text, are always preceded by a - (dash), and are often followed by arguments. In the previous example, -aggregate is an option, with name as its argument. An option and its arguments tell the program which entities to manipulate when running the command (for example, which aggregate, or which file system). In general, the issuer should provide the options for a command in the order detailed in the format description. The {} (braces separated by a vertical bar) indicate that the issuer must enter either one option or the other (-all or -aggregate in the previous example).

Command options are described in alphabetic order to make them easier to locate—this does not reflect the format of the command. The formats are presented the same as on your system.
Arguments
Arguments for options are highlighted in the text. The [ ] indicate that the issuer must enter either one argument or the other (-all or -aggregate in the preceding example). The ... (ellipsis) indicates that the issuer can enter multiple arguments.

Options
Some commands have optional, as well as required, options and arguments. Optional information is enclosed in [ ] (brackets). All options except -all or -aggregate in the previous example are optional.

Options
The following options are used with many zfsadm commands. They are also listed with the commands that use them.

-aggregate name
Specifies the aggregate name of the aggregate to use with the command.

-filesystem name
Specifies the file system to use with the command.

-help
Prints the online help for this command. All other valid options specified with this option are ignored. For complete details about receiving help, see "Receiving help" on page 152.

-size kbytes
Specifies the size in K-bytes for the kbytes argument.

-system sysname
Specifies the name of the system that the request will be sent to.

When an option is specified multiple times on one command, the first will be honored and the subsequent ones will be ignored. This can cause a subsequent argument to be interpreted as an option and be diagnosed as unrecognized.

Usage
Most zfsadm commands are administrative-level commands used by system administrators to manage file systems and aggregates. You can issue commands from OMVS, TSO/E, or as a batch job. Use the IOEZADM format for TSO/E and batch. For an example, see Figure 53 on page 158. The description of the zfsadm attach command shows an example of issuing them as a batch job. The other zfsadm commands can be run as a batch job in a similar manner.

For a batch job, the zfsadm options are specified in the EXEC PARM as a single subparameter (a single character string enclosed in apostrophes with no commas separating the options). You cannot put the ending apostrophe in column 72. If it needs go to the next line, use a continuation character in column 72 (continuing in column 16 with the ending apostrophe on the second line). Remember that a JCL EXEC PARM is limited to 100 characters. See the topic on the EXEC PARM in z/OS MVS JCL Reference.

zfsadm commands are serialized with each other. That is, when a zfsadm command is in progress, a subsequent zfsadm command is delayed until the active zfsadm completes. This also includes MOUNT of a compatibility mode aggregate (because an implicit attach occurs). This does not include zfsadm grow or implicit aggregate grow. zfsadm commands do not delay normal file system activity (except when the zfsadm command requires it, such as zfsadm quiesce).
zfsadm commands only work on zFS file systems and aggregates. All zfsadm commands work across sysplex members that are in a shared file system environment.

When supplying an argument to a zfsadm command, the option (for example -aggregate) associated with the argument (for example, OMVS.PRv.AGGR001.LDS0001) can be omitted if:

- All arguments supplied with the command are entered in the order in which they appear in the command’s syntax. (The syntax for each command appears with its description in this chapter.)
- Arguments are supplied for all options that precede the option to be omitted.
- All options that precede the option to be omitted accept only a single argument.
- No options, either those that accept an argument or those that do not, are supplied before the option to be omitted.
- The first option cannot be followed by an additional option before the vertical bar.

In the case where two options are presented in | (braces separated by a vertical bar), the option associated with the first argument can be omitted if that argument is provided; however, the option associated with the second argument is required if that argument is provided.

If it must be specified, an option can be abbreviated to the shortest possible form that distinguishes it from other options of the command. For example, the -aggregate option found in many zfsadm commands can typically be omitted or abbreviated to be simply -a. (One exception is the zfsadm attach command because it has an -aggrfull option.)

It is also valid to abbreviate a command name to the shortest form that still distinguishes it from the other command names in the suite. For example, it is acceptable to shorten the zfsadm grow command to zfsadm g because no other command names in the zfsadm command suite begin with the letter g. However, there are two zfsadm commands that begin with l: zfsadm lsaggr and zfsadm lsf. To remain unambiguous, they can be abbreviated to zfsadm lsa and zfsadm lsf.

The following examples illustrate three acceptable ways to enter the same zfsadm grow command:

- Complete command:
  
  ```
  zfsadm grow -aggregate omvs.prv.aggr001.lds0001 -size 50000
  ```

- Abbreviated command name and abbreviated options:
  
  ```
  zfsadm g -a omvs.prv.aggr001.lds0001 -s 50000
  ```

- Abbreviated command name and omitted options:
  
  ```
  zfsadm g omvs.prv.aggr001.lds0001 50000
  ```

The ability to abbreviate or omit options is intended for interactive use. If you imbed commands in a shell script, you should not omit options nor abbreviate them. If an option is added to a command in the future, it might increase the minimum unique abbreviation required for an existing option or change the order of options.

In general, zfsadm commands are processed on a worker thread while the zfsadm thread waits. If you cancel a zfsadm command that is taking a long time (for example, zfsadm grow or zfsadm config (to shrink a cache)), the zfsadm (waiting)
thread is cancelled, but the worker thread continues to process the request to completion. In addition, most `zfsadm` commands require a common `zfsadm` lock while they are processing. If the `zfsadm` command cannot get the lock, it waits for it to become available. This means, if you issue another `zfsadm` command (after canceling a previous one), it can be delayed by this common `zfsadm` lock, until the previous (possibly cancelled) command completes.

**Receiving help**

There are several different ways to receive help about `zfsadm` commands. The following examples summarize the syntax for the different help options available:

- **`zfsadm help`**
  - Displays a list of commands in a command suite.

- **`zfsadm help -topic`** _command_
  - Displays the syntax for one or more commands.

- **`zfsadm apropos -topic`** _string_
  - Displays a short description of any commands that match the specified _string_.

As of z/OS V1R13, when the `zfsadm` command displays help text or a syntax error message, it will show the name of the command as IOEZADM, instead of `zfsadm`. This occurs because the `zfsadm` command is not a binary module in the z/OS UNIX file system; rather, it is a shell script that invokes IOEZADM. IOEZADM is an entry that has the sticky bit on in the permissions. The sticky bit means that the IOEZADM module is found and executed from the user's STEPLIB, link pack area, or link list concatenation. (IOEZADM is usually located in SYS1.SIEALNKE.) However, you cannot run IOEZADM from the shell because IOEZADM is not normally in your PATH.

**Privilege required**

`zfsadm` commands that query information (for example, `lsfs`, `aggrinfo`) require the issuer to have READ authority to the data set that contains the IOEFSPRM file if you are using an IOEFSPRM file in your zFS PROC, or require no special authorization if you are using parmlib (IOEPRMxx). `zfsadm` commands that modify (for example `format`) additionally require that the issuer must have one of the following:

- **UID of 0.** If you are permitted READ to the BPX.SUPERUSER resource in the RACF FACILITY class, you can become a UID of 0 by issuing the `su` command.
- **READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.**

Specific privilege information is listed within each command’s description.

**Related information**

Commands:
- `zfsadm aggrinfo`
- `zfsadm apropos`
- `zfsadm attach`
- `zfsadm config`
- `zfsadm configquery`
- `zfsadm convert`
- `zfsadm define`
zfsadm

display

display

display

display

display

display

display

display

display

display

display

display

display

Files:
  IOEFSRPM

**zfsadm aggrinfo**

**Purpose**

Displays information about an aggregate, or all attached aggregates, if there is no specific aggregate specified.

**Format**

```bash
zfsadm aggrinfo [-aggregate name | -system sysname] [-fast | -long] [-level] [-help]
```

**Options**

- **-aggregate name**
  Specifies the name of an aggregate about which information is to be displayed. The aggregate must be attached. The aggregate name is not case-sensitive. It is translated to uppercase. If this option is omitted, information is provided about all of the attached aggregates on the system. Compatibility mode aggregates are implicitly attached when they are mounted.

- **-fast**
  Causes the command to display a single line of output for each attached aggregate. See "**Usage**" on page 154 for an explanation of the information that is displayed on each line.

- **-help**
  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- **-level**
  Prints the level of the **zfsadm** command. This option is useful when you are diagnosing a problem. Except for -help, all other valid options that are specified with -level are ignored.

- **-long**
  Causes the output of the command to be extended to display the following additional information about space usage in an aggregate:
  - Version of the aggregate
  - File system identification (auditfid)
  - Indicates sysplex-aware when the aggregate is sysplex-aware for read/write
zfsadm aggrinfo

- Indicates converttov5 if the aggregate has the converttov5 attribute
- Number of free 8-KB blocks
- Number of free 1-KB fragments
- Size of the log file
- Size of the filesystem table
- Size of the bitmap file
- If the aggregate is quiesced, the job name, system name and the time stamp of when the quiesce occurred.

-system sysname

Specifies the name of the system that owns the attached aggregates for which the information is displayed.

Usage

The zfsadm aggrinfo command lists information about the total amount of disk space and the amount of disk space currently available on attached aggregates. The -aggregate option can be used to specify a single aggregate about which information is to be displayed. If this option is omitted, information about all aggregates that are attached in the sysplex (if shared file systems are being used) or the system is displayed. In a shared file system environment, you can limit the display to a single system by using the -system option. Compatibility mode aggregates are implicitly attached when they are mounted.

This command displays a separate line for each aggregate. Each line displays the following information:

- The aggregate name.
- Whether the aggregate is read/write (R/W) or read-only (R/O), it is a mounted compatibility mode aggregate (COMP) or an attached compatibility mode aggregate (MULT), or the aggregate is currently quiesced (QUIESCED), disabled (DISABLED), or both.
- The amount of space available in KB.
- The total amount of space in the aggregate in KB. (To grow an aggregate using the zfsadm grow command, specify a number larger than this number.)
- If -long is specified, the version of the aggregate, the auditfid, sysplex-aware if the aggregate is sysplex-aware for read/write, the converttov5 attribute, the number of free 8-KB blocks, the number of free 1-KB fragments, the size of the log file, the size of the file system table, the size of the bitmap file, and if the aggregate is quiesced, the job name, timestamp, and system name of the job.

Privilege required

If you are using an IOEFSprm file in your ZFS PROC, the issuer must have READ authority to the data set that contains the IOEFSprm file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Example

Following is an example command that displays information about the disk space that is available on all aggregates that are attached in the sysplex.
zfsadm aggrinfo

DCEIMGKC:/DCEIMGKC/home/suimgkc> zfsadm aggrinfo -long
IOE200369I A total of 1 aggregates are attached to the sysplex.
PLEX.AGGR (R/W COMP QUIESCED): 559 K free out of total 720
version 1.5
auditfid C36C3F0 F0F3000E 0000
sysplex-aware, convertto5
  69 free 6k blocks; 7 free 1K fragments
  112 K log file; 16 K filesystem table
  8 K bitmap file
Quiesced by job SUIMGKC3 on system DCEIMGKC on Mon Feb 11 16:04:36 2013

Figure 52. Sample of zfsadm aggrinfo -long command

Related information

Commands
  zfsadm lsaggr

Files:
  IOEFSPRM

zfsadm apropos

Purpose

Shows each help entry containing a specified string.

Format

```
zfsadm apropos -topic string [-level] [-help]
```

Options

- **-help**  Prints the online help for this command. All other valid options specified with this option are ignored.

- **-level**  Prints the level of the zfsadm command. This is useful when you are diagnosing a problem. Except for -help, all other valid options specified with -level are ignored.

- **-topic**  Specifies the keyword string for which to search. If it is more than a single word, surround it with quotation marks (""") or another delimiter. Type all strings for zfsadm commands in all lowercase letters.

Usage

The zfsadm apropos command displays the first line of the online help entry for any zfsadm command containing the string specified by -topic in its name or short description. To display the syntax for a command, use the zfsadm help command.

Privilege required

If you are using an IOEFSPRM file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.
zfsadm apropos

Results

The first line of an online help entry for a command lists the command and briefly describes its function. This command displays the first line for any zfsadm command where the string specified by -topic is part of the command name or first line.

Examples

The following command lists all zfsadm commands that have the word list in their names or short descriptions:

zfsadm apropos list

lsaggr: list aggregates
lsfs: list filesystem information

Related information

Commands
zfsadm help

zfsadm attach

Purpose

Attaches an aggregate to zFS without mounting the file system.

Note: zfsadm agrinfo displays an attached compatibility mode aggregate as MULT because it is not mounted.

Format

```
zfsadm attach {-aggregate name [-system sysname]} 
[-aggrfull threshold,increment] 
[{-R/O | -ro | -rw}] [-nbs | -nonbs] [-aggrgrow | -noaggrgrow][-level] [help]
```

Options

- **-aggregate name**
  Specifies the name of the aggregate to be attached. The aggregate name is not case-sensitive. It is translated to uppercase. This aggregate does not need an entry in the IOEFSPRM file.

  Compatibility mode aggregates do not need to be attached with the zfsadm attach command. They are automatically attached on MOUNT of the compatibility mode file system.

- **-aggrfull threshold,increment**
  Specifies the threshold and increment for reporting aggregate full error messages to the operator. Both numbers must be specified. The first number is the threshold percentage and the second number is the increment percentage. For example, if 90,5 were specified, the operator is notified when the aggregate is 90% full, then again at 95% full, and again at 100% full. The default is the global aggrfull entry of the IOEFSPRM file.

- **-aggrgrow**
  Specifies that the aggregate should be dynamically grown if it runs out of
zfsadm attach

physical space. The aggregate (that is, the VSAM linear data set) must have a secondary allocation specified and there must be space available on the volume. The default is the aggrgrow option of the IOEFSPRM file.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

-level Prints the level of the zfsadm command. This is useful when you are diagnosing a problem. Except for -help, all other valid options that are specified with -level are ignored.

-nbs Specifies that new block security is used for file systems in this aggregate. New block security refers to the guarantee made when a system fails. If a file was being extended or new blocks were being allocated for the file, but the user data had not yet made it to the disk when the failure occurred, zFS shows the newly allocated blocks as all binary zeros and not whatever was on disk in those blocks at time of failure.

-nonbs The NONBS option is no longer supported; if NONBS is specified, it is ignored. zFS always runs with NBS on.

-noaggrgrow Specifies that the aggregate should not be dynamically grown if it runs out of physical space. The default is the aggrgrow option of the IOEFSPRM file.

-R/O |-ro Specifies that the aggregate should be opened in read-only mode. The default is read/write unless -R/O or -ro is specified.

-rw Specifies that the aggregate should be opened in read/write mode. The default is read/write unless -R/O or -ro is specified.

-system sysname Specifies the name of the system that will be the zFS owner of the aggregate. The system name is not case-sensitive. It is translated to uppercase.

Usage

The zfsadm attach command attaches zFS aggregates on this system.

If the attach fails because log recovery is unsuccessful, you can run the ioefsutl salvage batch utility with the -verifyonly option on the aggregate to determine if there is an inconsistency. If so, use ioefsutl salvage to recover the aggregate and reissue the zfsadm attach command.

The zfsadm lsaggr command can be used to display a current list of all aggregates that are attached on this sysplex with the zFS owning system indicated, or this system when -system is used.

If the DASD volume containing the zFS aggregate being attached is read-only, you might receive message IOEZ00336I. This indicates that the zFS aggregate indicator cannot be set in the catalog (actually, in the VVDS on the volume). The zFS aggregate is successfully attached. DFSMSdss backup (DUMP) will not automatically quiesce and unquiesce the zFS aggregate because it cannot determine that the VSAM linear data set is a zFS aggregate. If the zFS aggregate can be attached with the DASD volume in read/write, the zFS aggregate indicator will be set.
You can determine if the zFS aggregate indicator is set by using IDCAMS LISTCAT ALL against the zFS aggregate and looking for the zFS indicator in the output.

Compatibility mode aggregates do not need to be separately attached because they are attached during MOUNT processing. However, if you want to issue a zfsadm command against a compatibility mode aggregate without mounting the aggregate, you can use the zfsadm attach command. You might attach an aggregate to delete a .bak file system that is contained in it.

Privilege required

The issuer must have READ authority to the data set that contains the IOEFSPRM file and is required to be logged in as root or to have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class. If you are not using IOEFSPRM but instead, you are using parmlib (IOEPRMxx), the issuer is required to be logged in as root or to have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Examples

The following command attaches an aggregate.

```
zfsadm attach -aggregate OMVS.PRV.AGGR001.LDS0001
```

Figure 53 shows the same example as a job that invokes zfsadm attach.

```
//USERIDA JOB ,'Zfsadm Attach',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//AGGRINFO EXEC PGM=IOEZADM,REGION=0M,
// PARM=('attach -aggregate OMVS.PRV.AGGR001.LDS0001')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*
```

Figure 53. Job to attach an aggregate

If you want to specify the R/O option, you must specify a leading slash. Otherwise, Language Environment® will treat the characters before the slash as Language Environment parameters. That is, you must use PARM=('attach OMVS.PRV.AGGR001.LDS0001 -R/O').

Related information

Commands:

```
zfsadm lsgagr
```

File

IOEFSPRM
zfsadm config

**Purpose**

Changes the value of zFS configuration (IOEFSPRM) options in memory. See Chapter 12, “The zFS configuration options file (IOEPRMxx or IOEFSPRM),” on page 193 for a complete list of IOEFSPRM options.

**Format**

```
zfsadm config \[-admin_threads number\] \[-user_cache_size number[,fixed]\]
[\[-meta_cache_size number[,fixed]\] \[-log_cache_size number[,fixed]\]]
[-sync_interval number] \[-vnode_cache_size number\] \[-nbs (on|off)\]
[-fsfull threshold,increment] \[-aggrfull threshold,increment\]
[-trace_dsn POSE_dataset_name] \[-tran_cache_size number\]
[-msg_output_dsn Seq_dataset_name] \[-metaback_cache_size number[,fixed]\]
[-aggrgrow (on|off)] \[-romount_recovery (on|off)\]
[-convert_auditfid (on|off)] \[-client_reply_storage storage size\]
[-file_threads number] \[-client_cache_size cache size[,fixed]\]
[-meta_cache_size cache size] \[-sysplex_filesys_sharemode (rwshare|norwshare)\]
[-change_aggrversion_on_mount (on | off)]
[-format_aggrversion {4|5}] \[-converttov5 (on | off)\]
[-system sysname] \[-level\] \[-help\]
```

**Options**

When you change options that apply to zFS aggregates and file systems, the current default changes. However, the change does not affect file systems that have already been mounted until they have been unmounted and remounted. This includes:
- aggrfull
- aggrgrow
- convert_auditfid
- change_aggrversion_on_mount
- converttov5
- fsfull
- sysplex_filesys_sharemode

- **-admin_threads number**
  Specifies the number of threads defined to handle pfsctl or mount requests.

- **-aggrfull threshold,increment**
  Specifies the threshold and increment for reporting aggregate full error messages to the operator.

- **-aggrgrow on | off**
  Specifies whether an aggregate should be dynamically extended when it runs out of physical space.

- **-change_aggrversion_on_mount on | off**
  Specifies whether an aggregate should be changed to a version 1.5 aggregate on mount. The default is off.

- **-client_cache_size cache size[,fixed]**
  Specifies the size, in bytes, of the client cache. This is only meaningful when zFS is running sysplex-aware.

- **-client_reply_storage storage size**
  Specifies the number of bytes allocated for sysplex client reply storage. This is only meaningful when zFS is running sysplex-aware.
zfsadm config

-convert_auditfid on | off
  Specifies whether the zFS auditfid is automatically changed to the unique format on mount (attach). If on is specified, or defaulted, mount (attach) changes the standard auditfid format to the unique auditfid format if the mount (attach) is read/write. If off is specified (or the mount (attach) is read-only), the auditfid is unaffected.

-converttov5 on | off
  Specifies whether directories in a version 1.5 aggregate should be converted from v4 directories to extended (v5) directories as they are accessed. A version 1.4 aggregate is changed to a version 1.5 aggregate. You can override this setting at mount time by specifying CONVERTTOV5 or NOCONVERTTOV5.

-file_threads number
  Specifies the current number of file threads. This is only meaningful when zFS is running sysplex-aware.

-format_aggrversion 4 | 5
  Specifies whether a version 1.4 aggregate or a version 1.5 aggregate should be formatted by default.

-fsfull threshold,increment
  Specifies the threshold and increment for reporting file system full error messages to the operator.

-help
  Prints the online help for this command. All other valid options specified with this option are ignored.

-level
  Prints the level of the zfsadm command. This is useful when you are diagnosing a problem. Except for -help, all other valid options specified with -level are ignored.

-log_cache_size number [,fixed]
  Specifies the size, in bytes, of the cache used to contain buffers for log file pages. The fixed option reserves real storage for usage by zFS only.

-meta_cache_size number [,fixed]
  Specifies the size, in bytes, of the cache used to contain meta data. The fixed option reserves real storage for usage by zFS only.

-metaback_cache_size number [,fixed]
  Specifies the size of the backing cache for meta data. The fixed option reserves real storage for usage by zFS only.

-msg_output_dsn Seq_dataset_name
  Specifies the name of a data set that contains any output messages that come from the zFS PFS.

-nbs on | off
  Controls the global new block security. zFS always runs with new block security on. The off option is no longer supported; if specified, it is accepted but not used.

-romount_recovery on | off
  Specifies whether zFS will automatically avoid a read-only mount failure (zFS reason code EFxx6271) because of the need to run log recovery for this aggregate. This can occur when the aggregate has been mounted read/write and a failure occurred before it was unmounted. If the next mount is for read-only, log recovery needs to run for the mount to be successful. If on is specified and this situation occurs, zFS temporarily
mounts the aggregate read/write to allow log recovery to run and then zFS will unmount and then mount the aggregate read-only.

**-sync_interval number**
Specifies the number of seconds between the times where zFS flushes data in its buffers to disk. The default is 30 seconds.

**-sysplex_filesys_sharemode rwshare | norwshare**
Specifies the default for the mount PARM when a zFS read/write file system is mounted on a sysplex=filesys system. You can override this setting at mount time by specifying an alternate value in the actual mount PARM.

**-system sysname**
Specifies the name of the system that the configuration option change request will be sent to.

**-token_cache_size cache size**
Specifies the token cache size maximum. When the token_cache_size is decreased, it is really the maximum size that is being decreased. This is only possible if the current usage is less than the maximum size. The token cache size cannot be decreased to lower than the current usage. The current usage is displayed through the MODIFY ZFS,QUERY,STKM command. This is only meaningful when zFS is running sysplex-aware.

**-trace_dsn PDSE_dataset_name**
Specifies the name of a data set that contains the output of any operator MODIFY ZFS,TRACE,PRINT commands or the trace output if zFS abends.

**-tran_cache_size number**
Specifies the number of transactions in the transaction cache.

**-user_cache_size number [fixed]**
Specifies the size, in bytes, of the cache used to contain file data. The fixed option reserves real storage for usage by zFS only.

**-vnode_cache_size number**
Specifies the number of vnodes that will be cached by zFS.

**Usage**

The **zfsadm config** command changes the configuration options (in memory) that were specified in the IOEFSPRM file (or defaulted). The IOEFSPRM file is not changed. If you want the configuration specification to be permanent, you must modify the IOEFSPRM file because zFS reads the IOEFSPRM file to determine the configuration values when zFS is started. The values that can be specified for each option are the same as the values that can be specified for that option in the IOEFSPRM file. You can specify that the configuration option change request should be sent to another system by using the **-system** option. The following options cannot be set using the **zfsadm config** command:
- **-cmd_trace**
- **-debug_dsn**
- **-group**
- **-msg_input_dsn**
- **-trace_table_size**
- **-sysplex_state**
Privilege required

The issuer must have READ authority to the data set that contains the IOEFSRM file and must be root or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class. If you are not using IOEFSRM but instead, you are using parmlib (IOEPRMxx), the issuer is required to be logged in as root or to have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Examples

The following example changes the size of the user cache:
```
zfsadm config -user_cache_size 64M
```
IOEZ00300I Successfully set -user_cache_size to 64M

Related information

Command:
```
zfsadm configquery
```

File:
IOEFSRM

zfsadm configquery

Purpose

The `zfsadm configquery` command queries the current value of zFS configuration options.

Format

```
zfsadm configquery [-system sysname] [-adm_threads] [-aggrfull] [-aggrgrow]
[ -all] [-change_aggrversion_on_mount] [-client_cache_size] [-client_reply_storage]
[-format_aggrversion] [-fsfull] [-group] [-log_cache_size] [-meta_cache_size]
[-metaback_cache_size] [-msg_input_dsn] [-msg_output_dsn] [-nbs] [-romount_recovery]
[-sync_interval] [-syslevel] [-sysplex_filesys_sharemode] [-sysplex_state]
[ -token_cache_size] [-trace_dsn] [-trace_table_size] [-tran_cache_size]
[ -user_cache_size] [-vnode_cache_size] [-level] [-help]
```

Options

- `-adm_threads`
  Displays the number of threads that are defined to handle pfsctl or mount requests.

- `-aggrfull`
  Displays the threshold and increment for reporting aggregate full error messages to the operator.

- `-aggrgrow`
  Displays whether an aggregate should be dynamically extended when it runs out of physical space.

- `-all`
  Displays the full set of configuration options.
zfsadm configquery

-change_aggrversion_on_mount
  Displays whether a version 1.4 aggregate should be changed to a version 1.5 aggregate when it is mounted.

-client_cache_size
  Displays the size, in bytes, of the client cache. This is only meaningful when zFS is running sysplex-aware.

-client_reply_storage
  Displays the number of bytes allocated for sysplex client reply storage. This is only meaningful when zFS is running sysplex-aware.

-cmd_trace
  Displays whether command tracing is active.

-converttov5
  Displays whether an aggregate should be assigned the converttov5 attribute on mount or remount. This attribute controls whether v4 directories will be converted to extended (v5) directories as they are accessed.

-convert_auditfid
  Displays whether the zFS auditfid is automatically changed to the unique format on mount (attach). If on is specified or defaulted and the mount (attach) is read/write, the mount (attach) changes the standard auditfid format to the unique auditfid format. If off is specified or the mount (attach) is read-only, the auditfid is unaffected.

-debug_dsn
  Displays the name of the debug input parameters data set.

-file_threads
  Displays the current number of file threads. This is only meaningful when zFS is running sysplex-aware.

-format_aggrversion
  Displays whether an aggregate formatting default should be a to format as a version 1.4 or 1.5 aggregate.

-fsfull
  Displays the threshold and increment for reporting file system full error messages to the operator.

-group
  Displays the XCF group that is used by zFS for communication between sysplex members.

-help
  Prints the online help for this command. All other valid options that are specified with this option are ignored.

-level
  Prints the level of the zfsadm command. This is useful when you are diagnosing a problem. Except for -help, all other valid options specified with -level are ignored.

-log_cache_size
  Displays the size, in bytes, of the cache that is used to contain buffers for log file pages.

-meta_cache_size
  Displays the size, in bytes, of the cache that is used to contain metadata.

-metaback_cache_size
  Displays the size of the backing cache for metadata.
Displays the name of the data set that contains translated zFS messages.

Displays the name of a data set that contains any zFS initialization output messages that come from the zFS PFS.

Controls the global new block security. zFS always runs with new block security on. If you use `zfsadm config` to set `-nbs` to off, it is displayed as off, but the value is not used.

Displays whether read-only mount recovery is on or off. When `romount_recovery=on`, zFS temporarily mounts the aggregate read/write to allow log recovery to run, and then zFS unmounts and mounts the aggregate again in read-only format.

Displays the number of seconds in the interval that zFS flushes data in the buffers to disk.

Displays the zFS kernel (the PFS) information, including:
- The version and release of z/OS
- The service level and FMID of zFS
- The date and time the PFS was built
- Whether the PFS is running sysplex-aware on a file system basis (referred to as filesys), or sysplex-aware on a system basis (referred to as file), or not sysplex-aware (referred to as admin-only), and the zFS XCF protocol level when running in a shared file system environment. (For information about the XCF protocol level, see “Determining the XCF protocol interface level” on page 102.) When filesys is indicated, the default mount PARM (NORWSHARE or RWSHARE) also displays.

This is the same information that is displayed by the operator command `MODIFY ZFS,QUERY,LEVEL`. In contrast, `zfsadm configquery -level` shows the level information for the `zfsadm` command itself.

Displays the current default for the mount PARM (RWSHARE or NORWSHARE). It is only meaningful on systems running zFS `sysplex=filesys`.

Displays the sysplex state of zFS.

zFS is running in a sysplex-aware environment with `sysplex=filesys`.

Specifies the name of the system the report request is sent to retrieve the data requested.

Displays the current `token_cache_size` maximum. The current usage is displayed through the `MODIFY ZFS,QUERY,STKM` command. This is only meaningful when zFS is running sysplex-aware.

Displays the name of the data set that contains the output of any operator `modify zfs,trace,print` commands or the trace output if zFS abends.
**zfsadm configquery**

- **trace_table_size**
  Displays the size, in bytes, of the internal trace table.

- **tran_cache_size**
  Displays the number of transactions in the transaction cache.

- **user_cache_size**
  Displays the size, in bytes, of the cache that is used to contain file data.

- **vnode_cache_size**
  Displays the number of vnodes that will be cached by zFS.

**Usage**

The `zfsadm configquery` command displays the current value of zFS configuration options. The value is retrieved from zFS address space memory rather than from the IOEFSPRM file. You can specify that the configuration option query request should be sent to another system by using the `-system` option.

Ignore the following values when zFS is running non-sysplex aware. No storage is obtained even though a value might be reported.
- client_cache_size
- client_reply_storage
- file_threads
- token_cache_size

**Privilege required**

If you are using an IOEFSPRM file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

**Examples**

The following example displays the current value of the `user_cache_size` option:

```
zfsadm configquery -user_cache_size
```

IOEZ00317I The value for config option -user_cache_size is 64M.

To display all the zFS configuration options from each member, you can use the following command:

```
for sys in `$(zfsadm lssys | grep -v IOEZ00361I); do; echo; echo $sys; zfsadm configquery -all -system $sys; done
```

**Related information**

Command:
- `zfsadm config`

File:
- IOEFSPRM

---

**zfsadm convert**

**Purpose**

Converts a v4 directory that is contained in a read/write mounted version 1.5 aggregate to an extended (v5) directory. The aggregate is changed from a version 1.4 aggregate to a version 1.5 aggregate, if necessary. It can also be used to change
zfsadm convert

a version 1.4 aggregate to a version 1.5 aggregate without converting any directories.

Format

```
zfsadm convert {-path name | -aggrversion name} [-level] [-help]
```

Options

- **-aggrversion name**
  Specifies the aggregate name that should be changed from a version 1.4 aggregate to a version 1.5 aggregate. No directories are converted. The aggregate name is not case-sensitive. It is translated to uppercase.

- **-help**
  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- **-level**
  Prints the level of the `zfsadm` command. This is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

- **-path name**
  Specifies the path name of a directory that should be converted to an extended (v5) directory. The aggregate is changed to a version 1.5 aggregate first, if necessary.

Usage

The `zfsadm convert` command can be used to explicitly convert a v4 directory to an extended (v5) directory that is contained in a read/write mounted version 1.5 aggregate. In this case, the `-path` option is used. If the containing aggregate is a version 1.4 aggregate, the command will attempt to change the aggregate to a version 1.5 aggregate before converting the directory.

It can also be used to explicitly change a version 1.4 aggregate to a version 1.5 aggregate without converting any directories. In this case, the `-aggrversion` option is used.

The `zfsadm convert` command might cause the file system to grow if it needs more space for the extended (v5) directory.

The command must be issued from a z/OS V2R1 system and the zFS file system must be zFS owned on a z/OS V2R1 system. The aggregate must be mounted read/write.

Do not use this command before you have migrated all your systems to z/OS V2R1. If there are systems that are prior to z/OS V2R1 active in the shared file system environment, no conversion of a directory nor change of aggregate version takes place.

Privilege required

If you are using an IOEFSPRM file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.
Zfsadm convert

The issuer must be the owner of the directory and must have write permission (w) to the directory. If the aggregate version is to be changed, the issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Examples

The following example contains the steps to convert an existing version 1.4 aggregate to a version 1.5 aggregate, and to convert a v4 directory to an extended (v5) directory.

1. To display the version of the aggregate:

   ```
   # zfsadm aggrinfo PLEX.JMS.AGGR009.LDS0009 -long
   PLEX.JMS.AGGR009.LDS0009 (R/W COMP): 1271 K free out of total 1440
   version 1.4
   auditfid C3C6C3F0 F0F200A2 0000
   158 free 8k blocks; 7 free 1K fragments
   112 K log file; 16 K filesystem table
   8 K bitmap file
   ```

2. To change the version to 1.5:

   ```
   # zfsadm convert -aggrversion PLEX.JMS.AGGR009.LDS0009
   IOEZ00810I Successfully changed aggregate PLEX.JMS.AGGR009.LDS0009 to version 1.5.
   ```

3. To verify the aggregate version change:

   ```
   # zfsadm aggrinfo PLEX.JMS.AGGR009.LDS0009 -long
   PLEX.JMS.AGGR009.LDS0009 (R/W COMP): 1271 K free out of total 1440
   version 1.5
   auditfid C3C6C3F0 F0F200A2 0000
   158 free 8k blocks; 7 free 1K fragments
   112 K log file; 16 K filesystem table
   8 K bitmap file
   ```

4. To display the version of a directory:

   ```
   # zfsadm fileinfo /service9
   path: /service9
   *** global data ***
   fid 1,1  anode 69,516
   length 8192  format BLOCKED
   1K blocks 8  permissions 755
   uid,gid 0,10  access acl 0,0
   dir model acl 0,0  file model acl 0,0
   user audit F,F,F  auditor audit N,N,N
   set sticky,uid,gid 0,0,0  seclabel none
   object type DIR  object linkcount 3
   object genvalue 0x00000000  dir version 4
   dir name count 3  dir data version 1
   dir tree status VALID  dir conversion na
   file format bits na,na,na  file charset id na
   file cver na  charspec major,minor na
   direct blocks 25
   indirect blocks none
   refcount none
   ```

5. To convert the directory to an extended (v5) directory:

   ```
   # zfsadm convert -path /service
   IOEZ00791I Successfully converted directory /service9 to version 5 format.
   ```
6. To display the version of the directory again:

```
# zfsadm fileinfo /service9
path: /service9
*** global data ***
fid 1,1 anode 69,516
length 8192 format BLOCKED
1K blocks 8 permissions 755
uid,gid 0,10 access acl 0,0
dir model acl 0,0 file model acl 0,0
user audit F,F,F auditor audit N,N,N
set sticky,uid,gid 0,0,0 seclabel none
object type DIR object linkcount 3
object genvalue 0x00000000 dir version 5
dir name count 3 dir data version 1
dir tree status VALID dir conversion na
file format bits na,na,na file charset id na
file cver nacharspec major,minor na
direct blocks 25
indirect blocks none

Related information

Command:
zfsadm config

File:
IOEFSPRM

zfsadm define

Purpose

Defines a VSAM linear data set that can be formatted as a zFS aggregate.

Format

```
zfsadm define -aggregate name [-dataclass SMS_data_class]
[-managementclass SMS_management_class]
[-storageclass SMS_storage_class] [-catalog catalog]
[-system sysname] [-model model [catalog]]
[-volumes volume [volume ...]]
[-cylinders primary [secondary]] [-kilobytes primary [secondary]]
[-megabytes primary [secondary]] [-tracks primary [secondary]]
[-records primary [secondary]] [-level] [-help]
```

Options

-**aggregate name**
  Specifies the aggregate name of the aggregate to be defined. This will be
  the name of the VSAM linear data set that is defined. The aggregate name
  is not case-sensitive. It is translated to uppercase.

-**catalog catalog**
  Specifies the name of the catalog in which the VSAM linear data set is to
  be defined.

-**cylinders primary [secondary]**
  Specifies the primary and optionally, the secondary allocation size for the
VSAM linear data set in cylinders. The VSAM linear data set must have a secondary allocation size specified, if you want to use dynamic grow. See “Dynamically growing a compatibility mode aggregate” on page 28 for more information.

-dataclass SMS_data_class
  Specifies the name of the data class to be used when the VSAM linear data set is defined.

-help
  Prints the online help for this command. All other valid options specified with this option are ignored.

-kilobytes primary [secondary]
  Specifies the primary and optionally, the secondary allocation size for the VSAM linear data set in kilobytes. The VSAM linear data set must have a secondary allocation size specified, if you want to use dynamic grow. See “Dynamically growing a compatibility mode aggregate” on page 28 for additional information.

-level
  Prints the level of the zfsadm command. This is useful when you are diagnosing a problem. Except for -help, all other valid options specified with -level are ignored.

-managementclass SMS_management_class
  Specifies the name of the management class to be used when the VSAM linear data set is defined.

-megabytes primary [secondary]
  Specifies the primary and optionally, the secondary allocation size for the VSAM linear data set in megabytes. The VSAM linear data set must have a secondary allocation size specified, if you want to use dynamic grow. See “Dynamically growing a compatibility mode aggregate” on page 28 for additional information.

-model model [catalog]
  Specifies the name of the model and optionally, the model entry’s catalog to be used when the VSAM linear data set is defined.

-records primary [secondary]
  Specifies the primary and optionally, the secondary allocation size for the VSAM linear data set in records. When records is specified, the record size is assumed to be 4089 bytes. The VSAM linear data set must have a secondary allocation size specified, if you want to use dynamic grow. See “Dynamically growing a compatibility mode aggregate” on page 28 for additional information.

-storageclass SMS_storage_class
  Specifies the name of the storage class to be used when the VSAM linear data set is defined.

-system sysname
  Specifies the name of the system that the define request will be sent to.

-tracks primary [secondary]
  Specifies the primary and optionally, the secondary allocation size for the VSAM linear data set in tracks. The VSAM linear data set must have a secondary allocation size specified, if you want to use dynamic grow. See “Dynamically growing a compatibility mode aggregate” on page 28 for additional information.

-volumes volume
  Specifies the volume on which the VSAM linear data set can have space.
**zfsadm define**

**Usage**

The `zfsadm define` command defines a VSAM linear data set. The VSAM linear data set is available to be formatted as a zFS aggregate. The command creates a DEFINE CLUSTER command string for a VSAM linear data set with SHAREOPTIONS(3) and passes it to the IDCAMS utility. If a failure occurs, the `zfsadm define` command can display additional messages from IDCAMS indicating the reason for the failure.

**Privilege required**

The issuer of the `zfsadm define` command requires sufficient authority to create the VSAM linear data set.

**Examples**

The following command defines a VSAM linear data set.

```
zfsadm define -aggregate omvs.prv.aggr001.lds0001 -volumes prv000 prv001 -cylinders 10 5
```

**Related information**

Commands:

```
zfsadm format
```

---

**zfsadm delete**

**Purpose**

Removes a backup file system in a compatibility mode aggregate.

This command will be removed in a future release.

**Format**

```
zfsadm delete -filesystem name [-aggregate name] [-level] [-help]
```

**Options**

- `-aggregate name`
  
  Specifies the name of the aggregate where the zFS file system resides. It is specified to qualify the zFS file system name (-filesystem) when there are multiple zFS file systems with the same name in different aggregates. The aggregate name is not case-sensitive. It is always folded to uppercase.

- `-filesystem name`
  
  Specifies the name of the backup file system to be removed. Include the .bak extension. The file system name is case-sensitive.

- `-help`
  
  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- `-level`
  
  Prints the level of the `zfsadm` command. This is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.
zfsadm delete

Usage

The `zfsadm delete` command removes the backup zFS file system that is indicated by the `-filesystem` option from its aggregate. The aggregate containing the file system to be deleted must be attached. Removing a backup file system does not remove the read/write file system.

You can delete a compatibility mode file system (and its aggregate) by using the IDCAMS DELETE operation. This operation deletes the VSAM linear data set. For more information about renaming or deleting a compatibility mode aggregate, see “Renaming or deleting a compatibility mode aggregate” on page 37.

Privilege required

The issuer must have READ authority to the data set that contains the IOEFSPRM file and must be root or have READ authority to the SUPERUSER.FILESYSPFSCTL resource in the z/OS UNIXPRIV class. If you are instead using parmlib (IOEPRMxx), the issuer is required to be logged in as root or to have READ authority to the SUPERUSER.FILESYSPFSCTL resource in the z/OS UNIXPRIV class.

Examples

The following command deletes the backup (clone) file system from its attached compatibility mode aggregate:

```
zfsadm delete OMVS.USER.PAT.bak
```

IOEZ00105I File System OMVS.USER.PAT.bak deleted successfully

Related information

Commands:
- `zfsadm attach`
- `zfsadm detach`
- `zfsadm lsfs`

File:
- IOEFSPRM

zfsadm detach

Purpose

Detaches one or more aggregates from zFS. Any file systems contained in the detached aggregate are unavailable to zFS.

Format

```
zfsadm detach [[-aggregate aggregate name] [-all [-system sysname]]] [-level] [-help]
```
zfsadm detach

Options

-**aggregate aggregate name**
  Specifies the aggregate name of the aggregate to be detached. Use this option or use -all, but not both. The aggregate name is not case-sensitive. It is always translated to uppercase.

-**all**
  Specifies that all attached aggregates in the sysplex are to be detached. Use this option or use -aggregate but not both.

-**help**
  Prints the online help for this command. All other valid options specified with this option are ignored.

-**level**
  Prints the level of the zfsadm command. This option is useful when you are diagnosing a problem. Except for -help, all other valid options specified with -level are ignored.

-**system sysname**
  Specifies the name of the system where the aggregates to be detached reside. It cannot be specified without the -all option.

Usage

The zfsadm detach command is used to detach an aggregate. Detaching an aggregate makes it unavailable to the system. To detach one or more aggregates, use the -all or the -aggregate option to specify the aggregates to be detached. Use the -system option to limit the detach to a single system. The -system option cannot be specified without the -all option.

**zfsadm detach** does not detach mounted compatibility mode aggregates.

Privilege required

The issuer must have READ authority to the data set that contains the IOEFSPRM file and must be logged in as root or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class. If you are not using IOEFSPRM but instead, you are using parmlib (IOEPRMxx), the issuer is required to be logged in as root or to have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Examples

The following example shows a zfsadm detach command that detaches the aggregate OMVS.PRVS.AGGR001.LDS0001.

```bash
zfsadm detach -aggregate omvs.prvs.aggr001.lds0001
```

IOE00122I Aggregate OMVS.PRVS.AGGR001.LDS0001 detached successfully

Related information

Commands:
  zfsadm attach

Files:
  IOEFSPRM
zfsadm fileinfo

**Purpose**

Displays detailed information about a file or directory.

**Format**

```
zfsadm fileinfo -path name [{-globalonly | -localonly | -both}] [-level] [-help]
```

**Options**

- **-both**  
  Causes the command to display both global and local information about the file or directory.

- **-globalonly**  
  Causes the command to display global (on-disk) information about the file or directory. This option is the default.

- **-help**  
  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- **-level**  
  Prints the level of the `zfsadm` command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

- **-localonly**  
  Causes the command to display local (in memory on this system) information about the file or directory.

- **-path name**  
  Specifies the path name of a file or directory about which information should be displayed. The path name is case-sensitive.

**Usage**

The `zfsadm fileinfo` command can be used to display information about a file or directory. It supports files and directories in version 1.4 aggregates. It also supports files and v4 or extended (v5) directories in version 1.5 aggregates.

If an aggregate has the `converttov5` attribute assigned to it, accessing a v4 directory with `zfsadm fileinfo` can cause its conversion to an extended (v5) directory. For more information, see “Converting an existing v4 directory to an extended (v5) directory” on page 26.

The command must be issued from a z/OS V2R1 system. The file or directory must be contained in a file system that is locally zFS-owned or in a client file system.

Some of the fields are only applicable to files, some are only applicable to directories, some are only applicable to the local system and some are only applicable to client systems. There can also be attributes that are sometimes associated with a file or directory, such as ACLs. When these situations occur, the fields of the output display will contain values such as 0 or na or none, depending on the type of value that the field contains when it does have valid information.

If the `-globalonly` option is specified (or defaulted), the following fields are displayed:
**zfsadm fileinfo**

- **fid**  The inode and uniquifier separated by a comma.
- **anode**  Anode block and offset into anode block separated by a comma.
- **length**  Length of data (directories are multiples of 8 K).
- **format**  INLINE, FRAGMENTED, or BLOCKED.
- **1K blocks**  Number of blocks that are used to store data, in kilobytes.
- **permissions**  Permissions in octal format.
- **uid,gid**  UID and GID of owner that is separated by a comma.
- **access acl**  Anode index to ACL and length of ACL separated by a comma.
- **dir model acl**  Anode index to directory model ACL and length of ACL separated by a comma.
- **file model acl**  Anode index to file model ACL and length of ACL separated by a comma.
- **user audit**  User audit flags for read, write, and execute:
  - **N**  None
  - **S**  Audit successful attempts
  - **F**  Audit failed attempts
- **auditor audit**  Auditor audit flags for read, write, and execute:
  - **N**  None
  - **S**  Audit successful attempts
  - **F**  Audit failed attempts
- **set sticky,uid,gid**  Sticky bit, set uid, and set gid separated by a comma.
- **seclabel**  Security label for file or directory.
- **object type**  DIR or FILE or LINK or CHARSPEC.
- **object linkcount**  Link count for the object.
- **object genvalue**  Object general attributes. This value is taken from at_genvalue in the z/OS UNIX structure ATTR.
- **dir version**  The version of the directory; 5 indicates an extended (v5) directory and 4 indicates a v4 directory.
- **dir name count**  The number of objects in an extended (v5) directory.
- **dir data version**  A number that is incremented each time that the directory is changed.
dir tree status
For an extended (v5) directory, VALID (accessed by hash) or BROKEN (accessed as a flat file). Not applicable for a v4 directory.

dir conversion
For an extended (v5) directory, not applicable. For a v4 directory, FAILED (directory conversion was unsuccessful) or not applicable.

file format bits
For a file, the txt flag, the defer tag, the file format. For other objects, the text flag, the defer tag, and the file format are not applicable.

file charset id
The coded character set ID. This value is taken from at_charsetid in the z/OS UNIX structure ATTR.

file cver
Creation verifier. This value is taken from AT_cver in the z/OS UNIX structure ATTR.

charspec major,minor
Character special file, major number, minor number. Each character special file has a device major number, which identifies the device type, and a device minor number, which identifies a specific device of a given device type.

direct blocks
The block numbers of the first eight 8K blocks.

indirect blocks
The block numbers of the level 0, level 1 and level 2 trees.

mtime
Last modification time.

atime
Last access time.

ctime
Last change time.

create time
Create time.

reftime
Last reference time.

If the -localonly option is specified, the following fields are displayed:

vnode, vntok
Addresses of the ZFS vnode and the z/OS UNIX vnode.

opens

<table>
<thead>
<tr>
<th>ow</th>
<th>Number of tasks that are waiting to open due to deny mode opens</th>
</tr>
</thead>
<tbody>
<tr>
<td>oi</td>
<td>Number of internal opens</td>
</tr>
<tr>
<td>rd</td>
<td>Number of read opens</td>
</tr>
<tr>
<td>rw</td>
<td>Number of write opens</td>
</tr>
</tbody>
</table>

open deny

| rd   | Number of deny-read opens                                      |
| wr   | Number of deny-write opens                                     |
| ar   | Number of advisory deny-read opens                            |
| aw   | Number of advisory deny-write opens                           |

owner
zFS owning system.
file seq read
  Indicates if user file cache considers file to be read sequentially. Valid only for files.

file seq write
  Indicates if user file cache considers file to be written sequentially. Valid only for files.

file unscheduled
  Indicates the number of unscheduled pages (dirty data) in the user file cache for files.

file segments
  The number of 64-K segments of the file that is cached in the user file cache.

file dirty segments
  The number of dirty segments in the user file cache. Dirty segments are regions of the file that are either dirty and not yet written to disk, or are waiting for an I/O to disk to complete.

file meta issued
  Applicable to files or directories that were accessed by the sysplex client. It indicates if the client made a request recently to the server where the object's metadata was updated.

file meta pending
  Applicable to files or directories that are accessed by sysplex client. It indicates if the client has an outstanding request to the server where the object's metadata might be updated.

client cached fsp
  Indicates that the client has security information that is cached for the directory or file.

client cached anode
  Indicates that the client has the object’s attributes and location information for the directory or file.

client cached symlink
  Indicates that the content of a symbolic link was cached by the sysplex client. This flag is valid only for symbolic links.

client revoke
  Indicates if a revoke is in progress to this sysplex client for this file or directory.

client thrashing
  Indicates if the file or directory is considered thrashing by zFS, and as a result, uses the zFS thrash resolution interface to the server.

client token rights
  Indicates the token rights that are held by the sysplex client for the object.

client trash ops
  Number of forwarded requests.

client ops to server
  Number of requests that the client made to the server for this object.

client meta buffers
  Number of buffers in the metadata or backing cache for this object for the sysplex client.
client meta updates
Indicates whether the sysplex client has updated metadata for this object.

dirty meta buffers
For owners, indicates the number of dirty buffers in the metadata cache for this file or directory.

Privilege required
If you are using an IOEFSprm file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSprm file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

The issuer must have lookup authority (x) to the directory and READ authority (r) to the file.

Examples
The following example displays information for the /service9 directory:

```
zfsadm fileinfo -both /service9
path: /service9
*** global data ***
fid 1,1 anode 69,516
length 8192 format BLOCKED
1K blocks 8 permissions 755
uid,gid 0,10 access acl 0,0
dir model acl 0,0 file model acl 0,0
user audit F,F,F auditor audit N,N,N
set sticky,uid,gid 0,0,0 seclabel none
object type DIR object linkcount 2
object genvalue 0x00000000 dir version 4
dir name count na dir data version 0
dir tree status na dir conversion na
file format bits na,na,na file charset id na
file cver na charspec major,minor na
direct blocks 107
indirect blocks none
reftime none
*** local data from system DCEIMGVM ***
vnode,vntok 0x00000000,,0x794C0900 0x00FF7CA0,,0x00000000
opens ow=0 oi=0 rd=0 wr=0
open deny rd=0 wr=0 ar=0 aw=0
owner DCEIMGVM file seq read na
file seq write na file unscheduled na
file pending na file segments na
file dirty segments na file meta issued na
file meta pending na client cached fsp na
client cached anode na client cached symlink na
client revoke na client thrashing na
client token rights na client thrash ops na
client ops to server na client meta buffers na
client meta updates na dirty meta buffers 0
```
zfsadm format

Purpose

Formats a VSAM linear data set as a zFS aggregate.

Format

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>

Options

-aggregate name

Specifies the name of the aggregate to be formatted. The aggregate name is not case-sensitive. It is translated to uppercase.

-compat

Specifies that the zFS aggregate should be formatted as a compatibility mode aggregate. That is, it should be formatted as an aggregate and then a zFS file system should be created in the aggregate. The zFS file system will have the same name as the aggregate. -compat is the default but is ignored.

-group {gid | name}

Specifies the group owner of the root directory of the file system. It can be specified as a z/OS group ID or as a GID. The default is the GID of the issuer of the zfsadm format command. If only -owner is specified, the group is that owner’s default group.

-grow blocks

Specifies the number of 8-KB blocks that zFS uses as the increment for extension when the -size option specifies a size greater than the primary allocation.

-help

Prints the online help for this command. All other valid options that are specified with this option are ignored.

-initialempty blocks

This option is being allowed for compatibility with earlier versions and is ignored. One 8-KB block at the beginning of the aggregate is reserved for IBM use.

-level

Prints the level of the zfsadm command. This option is useful when you are diagnosing a problem. Except for -help, all other valid options that are specified with -level are ignored.

-logsize blocks

Specifies the size in 8-KB blocks of the log. The valid range is from 13 to 16384 blocks (128 megabytes). The default is 1% of the aggregate size. This default logsize will never be smaller than 14 blocks and it will never be larger than 4096 blocks (32 megabytes). This size is normally sufficient. However, a small aggregate that is grown to be very large will still have a small log. You might want to specify a larger log if you expect the aggregate to grow very large.
-newauditfid
  Specifies that the aggregate should be formatted with the zFS auditfid and stored in the aggregate. This is the default.

-nonewauditfid
  Specifies that the aggregate should not be formatted with a zFS auditfid stored in it.

-owner {uid 1 name}
  Specifies the owner of the root directory of the file system. It can be specified as a z/OS user ID or as a UID. The default is the UID of the issuer of the zfsadm format command.

-perms number
  Specifies the permissions of the root directory of the file system. It can be specified as an octal number (for example, o755), as a hexadecimal number (for example, x1ED), or as a decimal number (for example, 493). The default is o755 (owner read/write-execute, group read-execute, and other read-execute.)

-overwrite
  Specifies that an existing zFS aggregate should be overlaid. All existing data is lost. Use this option with caution. This option is not usually specified.

-size blocks
  Specifies the number of 8-KB blocks that should be formatted to form the zFS aggregate. The default is the number of blocks that fits in the primary allocation of the VSAM linear data set. If a number less than the default is specified, it is rounded up to the default. If a number greater than the default is specified, a single extend of the VSAM linear data set is attempted after the primary allocation is formatted unless the -grow option is specified. In that case, multiple extensions of the amount that is specified in the -grow option are attempted until the -size is satisfied. Space must be available on the volume.

-system sysname
  Specifies the system that the format request will be sent to.

-version4
  Specifies that the aggregate should be a version 1.4 aggregate. See the Usage section for the default value that is used.

-version5
  Specifies that the aggregate should be a version 1.5 aggregate. See the Usage section for the default value that is used.

Usage

The aggregate version will be as specified if the -version4 or -version5 argument is used. If neither is used, then the default aggregate version will be obtained from the zFS PFS format_aggrversion setting. See "IOEFSPRM" on page 193 for a description of the format_aggrversion variable.

The zfsadm format command formats a VSAM linear data set as a zFS aggregate. All zFS aggregates must be formatted before use. The zfsadm format command requires the zFS PFS to be active on the system. The size of the aggregate is as many 8-KB blocks as fits in the primary allocation of the VSAM linear data set or as specified in the -size option. To extend it, use the zfsadm grow command. If -overwrite is specified, all existing primary and secondary allocations are
**zfsadm format**

formatted and the size includes all of that space, and the backup change activity flag is set.

**Privilege required**

The issuer of the `zfsadm format` command must meet one of the following authorization requirements:

- Have ALTER authority to the VSAM linear data set.
- Be UID 0.
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

If you are using an IOEFSRM file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

**Examples**

The following command formats the VSAM linear data set as a compatibility mode aggregate.

```bash
zfsadm format -aggregate omvs.prev.aggr001.lds0001 -owner usera -group audit -perms o750
```

**Related information**

Commands:

- `zfsadm define`

Files:

- IOEFSRM

---

**zfsadm grow**

**Purpose**

Makes the physical size of an aggregate larger.

**Format**

```
zfsadm grow -aggregate name -size kbytes [-level] [-help]
```

**Options**

- `-aggregate name`
  
  Specifies the name of the aggregate to be grown. The aggregate name is not case-sensitive. It is always translated to uppercase.

- `-help`

  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- `-level`

  Prints the level of the `zfsadm` command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options specified with `-level` are ignored.

- `-size kbytes`

  Specifies the new total size in kilobytes of the aggregate after the grow operation. The size is rounded up to a control area (CA). (A control area is
zfsadm grow

normally a cylinder or less and is based on the primary and secondary allocation units. See [z/OS DFSMS Using Data Sets](#) for more information about allocation size boundary.) If zero is specified, the secondary allocation size is used. The value that is specified cannot exceed the size of a single volume.

**Usage**

The `zfsadm grow` command attempts to extend the size of an aggregate when the size specified is greater than the current size of the aggregate or when the size is specified as zero. If the extend fails (for example, if there is no space on the volume, or if size zero is specified and there is no secondary allocation specified for the VSAM linear data set), the grow operation fails. If the size specified is less than or equal to the current size of the aggregate, no extend is attempted and the command successfully returns. An aggregate cannot be made smaller than its current size. In any case, if the aggregate’s high used value is less than the aggregate’s high allocated value, the aggregate will be formatted up to the high allocated value (making the high used value equal to the high allocated value). The current (formatted) size of an aggregate can be determined by using the `zfsadm aggrinfo` command. The high used value (HI-U-RBA) and the high allocated value (HI-A-RBA) can be determined by using the IDCAMS LISTCAT ALL command. For an explanation of the rules that apply to extending a VSAM linear data set, see [z/OS DFSMS Using Data Sets](#).

The size of the file system free space is increased by the amount of additional space available.

**Privilege required**

The issuer must have READ authority to the data set that contains the IOEFSPRM file and must be logged in as root or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class. If you are not using IOEFSPRM but instead, you are using parmlib (IOEPRMxx), the issuer is required to be logged in as root or to have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

**Examples**

The following command displays the online help entry for the `zfsadm grow` command:

```
zfsadm grow -help
```

Usage: `zfsadm grow -aggregate <name> -size <size in K bytes> [-level] [-help]`

**Related information**

Command:
```
zfsadm aggrinfo
```
zfsadm help

Format

```
zfsadm help [-topic command...] [-level] [-help]
```

Options

- **-help**  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- **-level**  Prints the level of the zfsadm command. This is useful when you are diagnosing a problem. Except for -help, all other valid options that are specified with -level are ignored.

- **-topic command**  Specifies each command whose syntax is to be displayed. Provide only the second part of the command name (for example, lsfs, not zfsadm lsfs). Multiple topic strings can be specified. If this option is omitted, the output provides a short description of all zfsadm commands.

Usage

The zfsadm help command displays the first line (name and short description) of the online help entry for every zfsadm command if -topic is not provided. For each command name specified with -topic, the output lists the entire help entry.

Usage: zfsadm help [-topic command...] [-level] [-help]

The online help entry for each zfsadm command consists of the following two lines:

- The first line names the command and briefly describes its function.
- The second line, which begins with Usage:, lists the command options in the prescribed order.

Use the zfsadm apropos command to show each help entry containing a specified string.

Privilege required

If you are using an IOEFSPRM file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

The following command displays the online help entry for the zfsadm lsfs command and the zfsadm lsaggr command:

```
zfsadm help -topic lsfs lsaggr
```

```
zfsadm lsfs: list filesystem information
Usage: zfsadm lsfs [-aggregate <aggregate name>] [{-fast |-long}] [-level] [-help]
zfsadm lsaggr: list aggregates
Usage: zfsadm lsaggr [-level] [-help]
```

Related information

Command:

zfsadm apropos
**Purpose**

`zfsadm lsaggr` lists all currently attached aggregates for zFS. The owning system is displayed in a shared file system (sysplex) environment.

**Format**

```
zfsadm lsaggr [-system name] [-level] [-help]
```

**Options**

- `-help`  Prints the online help for this command. All other valid options that are specified with this option are ignored.
- `-level`  Prints the level of the `zfsadm` command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.
- `-system name`  Specifies the name of the system that owns the attached aggregates to be displayed.

**Usage**

`zfsadm lsaggr` displays information about all attached aggregates.

`zfsadm lsaggr` displays a separate line for each aggregate. Each line displays the following information:

- The aggregate name.
- The name of the system that is the zFS owner of the aggregate. If the aggregate is unowned, `*UNOWNED` is displayed.
- The mode of the aggregate.
- The status of the aggregate (for example, QUIESCED, DISABLED, or both).

You can use the `zfsadm aggrinfo` command to display information about the amount of disk space available on a specific aggregate or on all aggregates on a system.

**Privilege required**

If you are using an IOEFSprm file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSprm file. If you are using parmlib (`IOEPRMxx`), the issuer does not need special authorization.

**Examples**

The following example shows that five aggregates are attached to the system or the sysplex when running in a shared file system environment.

```
zfsadm lsaggr
OMVS.PRV.AGGR004.LDS0004 JS00000 R/W
OMVS.PRV.AGGR003.LDS0002 JS00000 R/O
OMVS.PRV.AGGR003.LDS0001 JS00000 R/W
OMVS.PRV.AGGR002.LDS0002 JS00000 R/W
OMVS.PRV.AGGR001.LDS0001 JS00000 R/W
```
zfsadm lsaggr

Related information

Command:
zfsadm aggrinfo

File:
IOEFSVRM

zfsadm lsfs

Purpose

zfsadm lsfs lists all the file systems on a given aggregate or all attached aggregates.

Format

```
zfsadm lsfs [-aggregate name | -system sysname] [-fast | -long] [-level] [-help]
```

Options

- **-aggregate name**
  - Specifies an aggregate name that is used to retrieve file system information. The aggregate name is not case-sensitive. It is always translated to uppercase. If this option is not specified, the command displays information for all attached aggregates.

- **-fast**
  - Causes the output of the command to be shortened to display only the aggregate name.

- **-help**
  - Prints the online help for this command. All other valid options that are specified with this option are ignored.

- **-level**
  - Prints the level of the zfsadm command. This option is useful when you are diagnosing a problem. Except for -help, all other valid options that are specified with -level are ignored.

- **-long**
  - Causes the output of the command to be extended to display the following additional information about space usage in a file system: the allocation limit, the free space limit, the size of the inode table, the number of file requests, the version of the file system, the creation date and time, and the last update date and time.

- **-system sysname**
  - Specifies the name of the system that owns the aggregates that contain the file systems to be displayed.

Usage

The zfsadm lsfs command displays information about file systems in aggregates. The file systems do not need to be mounted. The zfsadm lsfs command displays the following information for a specified aggregate or all attached aggregates on a system or all attached aggregates in the sysplex:

- The total number of file systems that are contained in the aggregate.
- The file system’s name (with a .bak extension, if appropriate).
- The type (RW for read/write, or BK for backup).
- Whether it is mounted.
The allocation usage and the free space usage, in kilobytes.
Whether the file system is online.
Whether the backup is being deleted.
The total number of file systems online, offline, busy, and mounted appear at the end of the output for all file systems.

If `-fast` is specified, it only displays the file system names.

If `-long` is specified, the following information is displayed:
• Total number of file systems that are contained in the aggregate.
• File system’s name.
• File system’s ID.
• The type (RW for read/write, or BK for backup).
• Whether it is mounted or not.
• State vector of the file system.
• Whether the file system is online or not.
• Whether the backup is being deleted.
• Allocation limit and allocation usage.
• Free space limit and free space usage.
• Size of the Filesystem Inode Table and the number of file requests.
• Version of the aggregate.
• Day, date, and time when the file system was created.
• Day, date, and time when the contents of the file system were last updated.
• Total number of file systems online, offline, busy, and mounted appears at the end of the output for all file systems.

Privilege required

If you are using an IOEFSPRM file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

The following example displays information for the aggregate OMVS.PRV.AGGR001.LDS0001:
```
zfsadm lsfs -aggregate omvs.prv.aggr001.lds0001 -long
IOEZ00129I Total of 1 file systems found for aggregate OMVS.PRV.AGGR001.LDS0001
OMVS.PRV.FS1 100000,,5 RW (Not Mounted)  states 0x10010005 On-line
  4294967232 K alloc limit; 9 K alloc usage
  25000 K quota limit; 9 K quota usage
  8 K Filesystem Inode Table 0 file requests
version 1.4
Creation Thu Aug 9 17:17:03 2001
Last Update Thu Aug 9 17:17:03 2001

Total file systems online 1; total off-line 0; total busy 0; total mounted 0
```

Related information

Commands:
None
**zfsadm lssys**

**Purpose**

Shows the names of the members in a sysplex.

**Format**

```zfsadm lssys [-level] [-help]```

**Options**

- `-help`  Prints the online help for this command. All other valid options specified with this option are ignored.

- `-level`  Prints the level of the `zfsadm` command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options specified with `-level` are ignored.

**Usage**

The `zfsadm lssys` command displays the names of the members in a sysplex.

**Privilege required**

If you are using an IOEFSPRM file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

**Examples**

The command that follows shows the current list of system names in the XCF group for zFS.

```
zfsadm lssys
IOEZ00361I A total of 3 systems are in the XCF group for zFS
DCEIMGVM
DCEIMGVQ
DCEIMGVN
```

**Related information**

Commands:

- `zfsadm lsaggr`

---

**zfsadm query**

**Purpose**

Displays internal zFS statistics (counters and timers) maintained in the zFS Physical File System (PFS).
### Format

```
```

### Options

- **-dircache**
  Specifies that the directory cache counters report should be displayed. Beginning with z/OS V1R13 this option is not meaningful; the report shows zeros.

- **-help**
  Prints the online help for this command. All other valid options specified with this option are ignored.

- **-iobyaggregate**
  Specifies that the I/O count by aggregate report should be displayed.

- **-iobydasd**
  Specifies that the I/O count by Direct Access Storage Device (DASD) report should be displayed.

- **-ioccounts**
  Specifies that the I/O count report should be displayed.

- **-knpfs**
  Specifies that the kernel counters report should be displayed. This option only displays counters for PFS calls on the zFS owner. It does not display (a second set of) counters for PFS calls when this system is a zFS client.

- **-level**
  Prints the level of the `zfsadm` command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options specified with `-level` are ignored.

- **-locking**
  Specifies that the locking statistics report should be displayed.

- **-logcache**
  Specifies that the log cache counters report should be displayed.

- **-metacache**
  Specifies that the metadata cache counters report should be displayed.

- **-reset**
  Specifies the report counters should be reset to zero. Should be specified with a report type. The reset takes place after displaying the current values. For example, if you enter `zfsadm query -knpfs -reset`, the command returns the current values for the kernel counters report before resetting to zero.

- **-storage**
  Specifies that the storage report should be displayed.

- **-system sysname**
  Specifies the name of the system the report request will be sent to, to retrieve the data requested.

- **-trancache**
  Specifies that the transaction cache counters report should be displayed.

- **-usercache**
  Specifies that the user cache report should be displayed.
zfsadm query

-vnodecache
   Specifies that the vnode cache counters report should be displayed.

Usage

The zfsadm query command is used to display performance statistics maintained by the zFS Physical File System.

Privilege required

If you are using an IOEFSRPM file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEFSRPM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

The following example is one of the queries that displays performance statistics.

zfsadm query -iobyaggr

<table>
<thead>
<tr>
<th>DASD</th>
<th>PAV VOLSER</th>
<th>I0s Mode</th>
<th>Reads</th>
<th>K bytes</th>
<th>Writes</th>
<th>K bytes</th>
<th>Dataset Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFC000</td>
<td>1</td>
<td>R/W</td>
<td>13</td>
<td>92</td>
<td>7641</td>
<td>30564</td>
<td>PLEX.JMS.AGGR001.LDS0001</td>
</tr>
<tr>
<td>CFC000</td>
<td>1</td>
<td>R/O</td>
<td>9</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>PLEX.JMS.AGGR002.LDS0002</td>
</tr>
<tr>
<td>CFC000</td>
<td>1</td>
<td>R/W</td>
<td>26</td>
<td>188</td>
<td>4483</td>
<td>17952</td>
<td>PLEX.JMS.AGGR004.LDS0004</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>---------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>---------</td>
<td>--------------------------</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td>48</td>
<td>340</td>
<td>12124</td>
<td>48516</td>
<td>+TOTALS+</td>
</tr>
</tbody>
</table>

Total number of waits for I/O: 52
Average 1/O wait time: 3.886 (msecs)

Related information

Commands:
   zfsadm lsaggr

zfsadm quiesce

Purpose

Specifies that an aggregate and the file system contained in it should be quiesced.

Format

zfsadm quiesce {-all | -aggregate name} [-level] [-help]

Options

-aggregate name
   Specifies the name of the aggregate that is to be quiesced. The aggregate name is not case-sensitive. It is always translated to uppercase. An aggregate must be attached to be quiesced. All current activity against the aggregate is allowed to complete but no new activity is started. Any mounted file systems are quiesced.

-all
   Specifies that all attached aggregates are to be quiesced. Use this option or use -aggregate.
zfsadm quiesce

- **-help**  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- **-level**  Prints the level of the zfsadm command. This option is useful when you are diagnosing a problem. Except for -help, all other valid options that are specified with -level are ignored.

Usage

The zfsadm quiesce command is used to temporarily drain activity to the aggregate. During this time:

- The aggregate cannot be detached, or grown.
- No activity can occur against mounted file systems.
- If you attempt to unmount a quiesced compatibility mode aggregate, the attempt fails unless you specify unmount force.

The aggregate can be the target of lsaggr, aggrinfo, lsfs (file systems are indicated as busy). While at least one RWSHARE aggregate remains quiesced, message IOEZ00581E is displayed on the zFS owning system's console.

While an RWSHARE file system is quiesced, the command D OMVS,F displays QUIESCED in the PFS EXCP field.

The aggregate is typically quiesced prior to backing up the aggregate. After the backup is complete, the aggregate can be unquiesced.

Privilege required

The issuer must have READ authority to the data set that contains the IOEFSPRM file and must be logged in as root or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class. If you are not using IOEFSPRM but instead, you are using parmlib (IOEPRMxx), the issuer is required to be logged in as root or to have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Examples

The following command quiesces the aggregate OMVS.PRV.AGGR001.LDS0001.

```
zfsadm quiesce -aggregate omvs.prv.aggr001.lds0001
```

IOEZ00163I Aggregate OMVS.PRV.AGGR001.LDS0001 successfully quiesced

Related information

Commands:
```
zfsadm aggrinfo
zfsadm unquiesce
```

zfsadm setauditfid

**Purpose**

Sets (or resets) the zFS auditfid in the mounted aggregate.
**zfsadm setauditfid**

**Format**

```
zfsadm setauditfid -aggregate aggrname [-force | -old] [-level] [-help]
```

**Options**

- `-aggregate aggrname`
  Specifies the name of the aggregate whose auditfid is to be set. The aggregate must be attached (mounted). The aggregate name is not case-sensitive. It is always translated to uppercase.

- `-force`
  Specifies to change the auditfid to a new zFS auditfid. If the aggregate already contains the new form of the zFS auditfid that you want to change to a different new zFS auditfid (for example, if you copy an aggregate and then rename it, but keep the old aggregate), you must specify `-force` to avoid inadvertently changing the zFS auditfid.

- `-help`
  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- `-level`
  Prints the level of the `zfsadm` command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

- `-old`
  Specifies that the zFS auditfid is set to binary zeros.

**Usage**

The `zfsadm setauditfid` command sets or resets the zFS auditfid in the aggregate on disk (based on the VOLSER and the cylinder, cylinder, head, head [CCHH] of the first extent of the aggregate). The aggregate must be attached (mounted). If neither `-force` nor `-old` are specified, a standard form auditfid (binary zeros) is changed to the unique form auditfid. If the aggregate already contains the unique form of the zFS auditfid and you want to change it to a different unique zFS auditfid (for example, if you copy an aggregate and then rename it - keeping the old one), you must specify `-force` to avoid inadvertently changing the zFS auditfid. The zFS auditfid is based on the VOLSER and the CCHH of the first extent, unless you specify `-old`. In that case, the zFS auditfid is set to binary zeros.

In a shared file system environment, whether the `zfsadm setauditfid` command is issued from the system owning the zFS aggregate or from a client system, the new auditfid value will only be visible on the zFS owning system. To make it visible on client systems, issue a remount to the same mode.

**Privilege required**

If you are using an IOEPSPRM file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOEPSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

The user must be UID 0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

**Examples**

```
zfsadm setauditfid -aggregate OMVS.PRV.AGGR001.LDS0001 -force
```
Related information

Commands:
  zfsadm aggrinfo
  zfsadm format

File:
  IOEFSPRM

**zfsadm unquiesce**

**Purpose**

The `zfsadm unquiesce` command makes an aggregate (and the file system that is contained in the aggregate) available to be accessed.

**Format**

```text
zfsadm unquiesce { -all | -aggregate name } [-level] [-help]
```

**Options**

- **-aggregate name**
  Specifies the name of the aggregate that is to be unquiesced. The aggregate name is not case-sensitive. It is always translated to uppercase. An aggregate must be attached to be unquiesced. All current activity against the aggregate is allowed to resume. Any mounted file systems are unquiesced.

- **-all**
  Specifies that all attached aggregates are to be unquiesced. Use this option or use `aggregate`.

- **-help**
  Prints the online help for this command. All other valid options that are specified with this option are ignored.

- **-level**
  Prints the level of the `zfsadm` command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

**Usage**

The `zfsadm unquiesce` command allows activity that was suspended by `zfsadm quiesce`, to be resumed.

The aggregate is typically quiesced prior to backing up the aggregate. After the backup is complete, the aggregate can be unquiesced and the backup change activity flag can be reset.

**Privilege required**

The issuer must have READ authority to the data set that contains the IOEFSPRM file and must be logged in as root or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class. If you are not using IOEFSPRM but instead, you are using parmlib (IOEPRMxx), the issuer is required to be logged in as root or to have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.
zfsadm unquiesce

Examples

The following command unquiesces the aggregate OMVS.PRV.AGGR001.LDS0001.

```
zfsadm unquiesce -aggregate omvs.prv.aggr001.lds0001
```

IOEZ00166I Aggregate OMVS.PRV.AGGR001.LDS0001 successfully unquiesced

Related information

Command:

```
zfsadm aggrinfo
zfsadm quiesce
```
Chapter 12. The zFS configuration options file (IOEPRMxx or IOEFSPRM)

This section describes the IOEFSPRM file, which is a data set that is used during zFS processing.

IOEFSPRM

Purpose

The IOEFSPRM file lists the configuration options for the zFS PFS and the batch utilities ioefsutl and ioeagslv. There is no mandatory information in this file; therefore, it is not required. The options all have defaults. However, if you need to specify any options (for tuning purposes, for example), you must have an IOEFSPRM file.

The location of the IOEFSPRM file can be specified by the IOEZPRM DD statement in the zFS PROC and in the JCL for the ioefsutl or ioeagslv batch utilities. However, the preferred method for specifying the zFS configurations options file is to use the IOEPRMxx parmlib member as described in “Using PARMLIB (IOEPRMxx)” on page 194. If you still want to use a single IOEFSPRM file, specify the IOEZPRM DD statement in your JCL. The IOEFSPRM file is typically a PDS member, so the IOEZPRM DD statement might look like the following example:

```
//IOEZPRM DD DSN=SYS4.PVT.PARMLIB(IOEFSPRM),DISP=SHR
```

If you need to have separate IOEFSPRM files and you want to share the zFS PROC in a sysplex, you can use a system variable in the zFS PROC so that it points to different IOEFSPRM files. The IOEZPRM DD might look like the following:

```
//IOEZPRM DD DSN=SYS4.PVT.&SYSNAME..PARMLIB(IOEFSPRM),DISP=SHR
```

Your IOEFSPRM file might reside in SYS4.PVT.SY1.PARMLIB(IOEFSPRM) on system SY1; in SYS4.PVT.SY2.PARMLIB(IOEFSPRM) on system SY2; and others.

If you want to share a single IOEFSPRM file, you can use system symbols in data set names in the IOEFSPRM file. For example, msg_output_dsn=USERA.&SYSNAME..ZFS.MSG60UT results in USERA.SY1.ZFS.MSG60UT on system SY1. Each system has a single (possibly shared) IOEFSPRM file.

Any line beginning with # or * is considered a comment. The text in the IOEFSPRM file is not case-sensitive. Any option or value can be uppercase or lowercase. Blank lines are allowed. Do not have any sequence numbers in the IOEFSPRM file. If you specify an invalid text value, the default value is assigned. If you specify an invalid numeric value, and it is smaller than the minimum allowed value, the minimum value is assigned. If you specify an invalid numeric value, and it is larger than the maximum allowed value, the maximum value is assigned.
Using PARMLIB (IOEPRMxx)

The preferred alternative to a IOEZPRM DDNAME is specifying the IOFSPRM file as a parmlib member. In this case, the member has the name IOEPRMxx, where xx is specified in the parmlib member list.

When the IOFSPRM is specified in a DD statement, there can only be one IOFSPRM file for each member of a sysplex. Using PARMLIB, zFS configuration options can be specified in a list of configuration parmlib files. This allows an installation to specify configuration options that are common among all members of the sysplex (for example, adm_threads) in a shared IOEPRMxx member and configuration options that are system-specific (for example, trace_dsn) in a separate, system-specific IOEPRMxx member. If a configuration option is specified more than once, the first one found is taken. For more information about PARMLIB, see z/OS MVS Initialization and Tuning Reference.

The IOEPRMxx files are contained in the logical parmlib concatenation. The logical parmlib concatenation is a set of up to ten partitioned data sets defined by parmlib statements in the LOADxx member of either SYSn.IPLPARM or SYS1.PARMLIB. The logical parmlib concatenation contains zFS IOEPRMyy members that contain zFS configuration statements. Columns 72-80 are ignored in the IOEPRMyy member. The yy values are specified in the PARM option of the FILESYSTYPE statement for the zFS PFS (in the BPXPRMxx). The only valid value that can be specified on the PARM option for the zFS PFS is the parmlib search parameter PRM=. The PARM string is case-sensitive. As the following example shows, you must enter the string in uppercase.

```
FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSOM)
  ASNAME(ZFS,'SUB=MSTR')
  PARM('PRM=(01,02,03)')
```

The parmlib concatenation can also be specified in the ioeagslv and ioefsut1 batch utility parameters. Specify the -PRM keyword in the PARM string on the EXEC statement to use IOEPRMxx parameter file members. For more information, see “ioeagslv” on page 129 and “ioefsut1” on page 134.

Up to 32 member suffixes can be specified. You can also use any system symbol that resolves to two characters.

```
FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSOM)
  ASNAME(ZFS,'SUB=MSTR')
  PARM('PRM=(01,&SYSCLONE.)')
```

See Figure 47 on page 134 for an example of using PRM.

If &SYSCLONE.=AB, this specifies that PARMLIB member IOEPRMAB should be searched after parmlib member IOEPRMO1. IOEPRM01 can contain common configuration options and IOEPRMAB can contain configuration options that are specific to system AB. If a parmlib member is not found, the search for the configuration option will continue with the next parmlib member.
To specify 32 members, type the member suffixes up to column 71; then, continue
them in column 1 on the next line, as shown in Figure 54.

If no PRM suffix list is specified (and no IOEZPRM DD is specified in their
respective JCL), then parmlib member IOEPRM00 is read. Parmlib support is only
used when no IOEZPRM DD is present in the JCL.

IOEFSPRM and IOEPRMxx

Descriptions of the valid configuration variables and their respective allowed
values follow. If no IOEFSPRM file is found, the default values for each
configuration value are used.

Usage

The following processing options are used for the zFS PFS.

adm_threads

Specifies the number of threads that are defined to handle pfsctl or mount
requests.

Default value: 10

Expected value: A number in the range 1 - 256.

Example: adm_threads=5

aggrfull

Specifies the threshold and increment for reporting aggregate utilization
messages to the operator. The aggrfull parameter is independent of
fsfull. However, aggrfull reports are based on free 8-K blocks; while
fsfull reports are based on free 1 K blocks. The aggrfull value tends to
give a more accurate view of free space and is the recommended choice.

Default value: Off

Expected value: Two numbers in the range 1 - 99 within parentheses that
are separated by a comma.

Example: aggrfull(90,5)

aggrgrow

Specifies whether aggregates can be dynamically extended when they
become full. As of z/OS V1R13, by default, a zFS read/write mounted file
system mounted on a system running z/OS V1R13 or later attempts to
dynamically extend when it runs out of space. The aggregate (that is, the
VSAM linear data set) must have a secondary allocation that is specified to
be dynamically extended and there must be space on the volume(s). This
global value can be overridden on the MOUNT command for compatibility
mode aggregates. For an explanation of the rules for extending a VSAM
LDS, see z/OS DFSMS Using Data Sets.

**Default value:** On

**Expected value:** On or Off

**Example:** aggrgrow=on

### change_aggrversion_on_mount

Specifies whether a version 1.4 aggregate should be changed to a version
1.5 aggregate on a primary read/write mount. No directories are converted
to extended (v5) directories. The CONVERTTOV5 or NOCONVERTTOV5
MOUNT PARM overrides this option.

**Default value:** Off

**Expected value:** On or Off

**Example:** change_aggrversion_on_mount=off

### client_cache_size

Specifies the amount of storage that is used to cache sysplex client user
data. You can also specify a fixed option that indicates the pages are
permanently fixed for performance. This option is in contrast to
user_cache_size (see client_cache_size). This option is only used for zFS
sysplex-aware read/write file systems that are owned on a release prior to
z/OS V1R13.

The fixed option reserves real storage for use by zFS only.

**Default value:** 32M

**Expected value:** A number in the range 10M - 65536M. K or M can qualify
the number.

**Example:** client_cache_size=256M

### client_reply_storage

Specifies the amount of storage that is used to handle sysplex server
replies.

**Default value:** 10 M

**Expected value:** A number in the range 2M - 128M. K or M can qualify the number.

**Example:** client_reply_storage=8M

### convert_auditfid

Specifies whether the zFS auditfid of an aggregate is automatically
converted from the old form auditfid (binary zeros) to the new form
auditfid on a read/write mount (attach). If the auditfid is already the new
form, it is not changed. An auditfid of the new form will cause zFS to
generate new auditids for files and directories in the file system.
Default value: On

Expected value: On or Off

Example: convert_auditfid=on

converttov5
Specifies whether a zFS read/write file system is assigned the converttov5 attribute. If it is assigned the converttov5 attribute and the aggregate is a version 1.5 aggregate, zFS will automatically convert directories from v4 to extended (v5) as they are accessed. If the converttov5 attribute is assigned at primary mount time, a version 1.4 aggregate will be changed to a version 1.5 aggregate. The CONVERTTOV5 or NOCONVERTTOV5 MOUNT PARM overrides this option.

If automatic directory conversion for a directory fails, it is not attempted again until the file system is unmounted and mounted again.

Default value: Off

Expected value: On or Off

Example: converttov5=off

file_threads
Specifies the number of threads that handle sysplex server requests.

Default value: 32

Expected value: A number in the range 1 - 256.

Example: file_threads=50

flc
This option will not be supported after z/OS V2R1. It specifies whether large FLC buffers are to be created. They are not created by default and specifications cannot be modified after zFS has initialized. Specifications for mindirsize, bufsize, bufnum and inactive values must be in this format, separated by commas:

flc=mindirsize,bufsize,bufnum,inactive

Default value: None

Expected value: One of the following values:

- mindirsize. The minimum size of a directory for which a Large FLC buffer will be used. A K or M can be appended to the value to mean kilobytes or megabytes, respectively.
- bufsize. The size of each buffer that will be used to hold the directory entries. A K or M can be appended to the value to mean kilobytes or megabytes, respectively.
- bufnum. The number of Large FLC buffers.
- inactive. The number of idle seconds before zFS considers Large FLC buffer to be inactive.

Example:

flc=256K,8M,30,45
flc=256K,8M,30,45
**format_aggrversion**
Specifies the default version of an aggregate when formatting it. Each method for formatting a zFS aggregate gets this value from the zFS PFS if no version is specified.

Default value: 4

Expected value: 4 (meaning format a version 1.4 aggregate) or 5 (meaning format a version 1.5 aggregate)

Example: format_aggrversion=4

**fsfull**
Specifies the threshold and increment for reporting file system utilization messages to the operator. The fsfull parameter is independent of aggrfull. Whereas aggrfull reports are based on free 8-KB blocks, fsfull reports are based on free 1-KB blocks. The aggrfull parameter tends to give a more accurate view of free space and is the recommended choice.

Default value: Off

Expected value: Two numbers in the range 1 - 99 within parentheses and separated by a comma.

Example: fsfull(85,5)

**group**
Specifies the XCF group that zFS uses to communicate between sysplex members. The Expected value characters must be acceptable to XCF. Generally, the characters A-Z, 0-9 and the national characters ($, # and @) are acceptable. The value that is specified must match on all systems in the sysplex that participate in a shared file system environment. Normally, there is no reason to specify this option. For more details, see the GRPNAME parameter of the IXCJOIN macro in z/OS MVS Programming: Sysplex Services Reference.

Default value: IOEZFS

Expected value: 1 to 8 characters

Example: group=IOEZFS1

**log_cache_size**
Specifies the size of the cache that is used to contain buffers for log file pages. You can also specify a fixed option, which indicates that the pages are permanently fixed for performance. The fixed option reserves real storage for usage by zFS only.

Default value: 16 M

Expected value: 1 to 8 characters

Example: log_cache_size=32M, fixed

**meta_cache_size**
Specifies the size of the cache used to contain metadata. You can also specify a fixed option, which indicates that the pages are permanently fixed for performance. The fixed option reserves real storage for usage by zFS only.
zFS provides a check to see if the sum of the metadata cache size and metadata backing cache size is less than the sum of the default metadata cache size and metadata backing cache size. For more information, see ZFS_VERIFY_CACHESIZE in IBM Health Checker for z/OS: User’s Guide.

**Default value:** If metaback_cache_size is specified, then meta_cache_size is 64 M. If metaback_cache_size is not specified, zFS calculates 10% of real storage that the system has available during ZFS initialization.

- If this amount is less than 64 M, then meta_cache_size is assigned 64 M and there is no metaback_cache created.
- If this amount is between 64 M and 100 M, then meta_cache_size is assigned 10% of real storage size and there is no metaback_cache created.
- If this amount is in the range 100 M to 2G+100M, then meta_cache_size is assigned 100 M and metaback_cache_size is assigned the remainder of the 10% of real storage size.
- If the amount is greater than 2G+100M, then meta_cache_size is assigned 100M and metaback_cache_size is assigned 2 G.

**Expected value:** A number in the range 1 M - 1024 M. A K or M can be appended to the value to mean kilobytes or megabytes, respectively.

**Example:** meta_cache_size=64M, fixed

**metaback_cache_size**

Specifies the size of the backing cache that is used to contain metadata. This resides in a data space and can optionally be used to extend the size of the metadata cache. You can also specify a fixed option, which indicates that the pages are permanently fixed for performance. Note, the fixed option reserves real storage for usage by zFS only.

zFS provides a check to see if the sum of the metadata cache size and metadata backing cache size is less than the sum of the default metadata cache size and metadata backing cache size. For more information, see ZFS_VERIFY_CACHESIZE in IBM Health Checker for z/OS: User’s Guide.

**Default value:** If meta_cache_size is specified then no metaback cache is created. Otherwise, see the default calculation description in meta_cache_size.

**Expected value:** A number in the range 1 M - 2048 M. A K or M can be appended to the value to mean kilobytes or megabytes, respectively.

**Example:** metaback_cache_size=64M, fixed

**quiesce_message_delay**

Specifies the minimum number of seconds to delay issuing the IOEZ00581E message after it is determined there is at least one quiesced aggregate and it needs to be displayed.

**Default value:** 30

**Expected value:** A number in the range 30 - 21474836.  

**Example:** quiesce_message_delay=300

**romount_recovery**

Specifies whether zFS will automatically avoid a read-only mount failure
because of the need to run log recovery for this aggregate. This can occur when the aggregate has been mounted read/write, and then a failure occurs before it was unmounted. If the next mount is for read-only, log recovery must run for the mount to be successful. When this situation occurs and `romount_recovery=on`, zFS temporarily mounts the aggregate read/write to run log recovery, and then zFS unmounts and mounts the aggregate read-only.

**Default value:** Off

**Expected value:** On or Off

**Example:** `romount_recovery=on`

### recovery_max_storage

Indicates the maximum amount of zFS address space storage to use for concurrent log recovery during multiple concurrent aggregate mounts (attaches). This allows multiple concurrent mounts to occur when sufficient storage is available for multiple concurrent log recovery processing.

**Default value:** 256 M

**Expected value:** A number in the range 128 M - 512 M.

**Example:** `recovery_max_storage=128M`

### sync_interval

Specifies the number of seconds between syncs.

**Default value:** 30

**Expected value:** A number in the range 11 - 21474836.

**Example:** `sync_interval=45`

### sysplex

Starting with z/OS V1R13, zFS always runs sysplex aware by file system, regardless of the sysplex specification. If you specify `sysplex=on`, zFS changes the default of `sysplex_filesys_sharemode` to `rwshare`. Otherwise, the default for `sysplex_filesys_sharemode` is `norwshare`. If you specify `sysplex=off`, the result is the same as specifying `sysplex=filesys`. For information about whether to make a read/write file system sysplex aware, see "Using zFS read/write sysplex-aware file systems" on page 15.

**Default value:** `filesys`

**Expected value:** Off, `filesys`, or `on`, if BPXPRMxx specifies `SYSPLEX(YES).`

Off, if BPXPRMxx does not specify `SYSPLEX(YES).`

**Tip:** Specify `sysplex=filesys`.

**Example:** `sysplex=filesys`

### sysplex_filesys_sharemode

Specifies the default for the mount PARM for a zFS read/write file system that is mounted on a `sysplex=filesys` system. For information about
whether to make a read-write file system sysplex aware, see “Using zFS read/write sysplex-aware file systems” on page 15.

Default value: norwshare (unless sysplex=on was specified, then the default is rwshare)

Expected value: rwshare or norwshare

Example: sysplex_filesys_sharemode=rwshare

token_cache_size
Specifies the maximum number of tokens in the server token manager cache to use for cache consistency between zFS members. The number of tokens initially allocated for the server token manager cache is 20480.

Default value: Double the number of vnodes (see vnode_cache_size) when running in a shared file system environment and sysplex-aware, or 20480 when running in a shared file system environment and non-sysplex aware, or <no value> otherwise (only meaningful when zFS is running sysplex-aware).

Expected value: A number in the range 20480 - 2621440.

Example: token_cache_size=30720

tran_cache_size
Specifies the initial number of transactions in the transaction cache.

Default value: 2000

Expected value: A number in the range 200 - 1000000

Example: tran_cache_size=4000

user_cache_size
Specifies the size, in bytes, of the cache that is used to contain file data. You can also specify a fixed option, which indicates that the pages are permanently fixed for performance. The fixed option reserves real storage for usage by zFS only.

zFS provides a check to see if the sum of the metadata cache size and metadata backing cache size is less than the sum of the default metadata cache size and metadata backing cache size. For more information, see ZFS_VERIFY_CACHESIZE in IBM Health Checker for z/OS: User’s Guide.

Default value: zFS calculates 10% of real storage the system has available during zFS initialization. If this amount is less than 256 M, then the default is 256 M. If this amount is between 256 M and 2 G, then the default is 10% of real storage. If the amount is greater than 2 G, then the default is 2 G.

Expected value: A number in the range 10 MB - 65536 MB (64 G). K or M can be appended to the value to mean kilobytes or megabytes.

Example: user_cache_size=64M,fixed

vnode_cache_size
Specifies the initial number of vnodes that will be cached by zFS. The number of vnodes with vnode extensions will not exceed this number.
IOEFSPRM

**Default value:** 32768 (will grow if z/OS UNIX needs more than this number)

**Expected value:** A number in the range 32 to 500000.

**Example:** vnode_cache_size=131072

The following options are used during debugging of the zFS PFS, the batch utilities (ioeagfmt, ioeagslv, and ioefsutl) and the zfsadm command. They might not apply to the utilities and commands listed in the preceding section.

**cmd_trace**
Specifies whether command tracing is done for the ioeagfmt batch utility or a zfsadm command. If On, a zFS trace will be printed in the dataset specified by the zFS PFS trace_dsn configuration option after the batch utility or command completes. A trace from ioeagfmt will have a member name of IOEAGT01. A trace from a zfsadm command will have a member name of ZFSADT01.

**Default value:** Off

**Expected value:** On or Off.

**Example:** cmd_trace=on

**debug_settings_dsn**
Specifies the name of a data set containing debug classes to enable when the zFS PFS or the batch utilities start. It is read when zFS is started (or restarted). The debug classes are also used by the batch utilities.

**Default value:** None

**Expected value:** The name of a data set containing debug classes to enable.

**Example:** debug_settings_dsn=usera.zfs.debug.input(file1)

**max_errors**
The maximum number of errors that the salvager program allows before it stops. If this limit is exceeded, the salvager program ends with message IOEZ00752E.

**Default value:** 100000

**Expected value:** A number in the range 1000 - 100000

**Example:** MAX_ERRORS=5000

**msg_input_dsn**
Specifies the name of a data set containing translated zFS messages. It is specified when the installation uses messages that are in languages other than English. (When you use English messages, do not specify this option.) It is read when zFS or the batch job is started (or restarted). Currently, Japanese messages are supported.

**Default value:** None

**Expected value:** The name of a data set containing translated zFS messages.
**Example:**  msg_input_dsn=usera.sioemjpn

**msg_output_dsn**

Specifies the name of a data set that contains any output messages that come from the zFS PFS during initialization. See [Chapter 8, “Performance and debugging,” on page 69.](#) This is not a required parameter.

**Default value:** None

**Expected value:** The name of a data set that contains zFS PFS messages issued.

**Example:**  msg_output_dsn=usera.zfs.msg.out

**trace_dsn**

Specifies the name of a data set that contains the output of any operator MODIFY ZFS,TRACE,PRINT commands or the trace output if the zFS PFS or the batch utilities abends. Each trace output creates a member in the PDSE. Traces that come from the zFS PFS kernel have member names of ZFSKNTnnn. Traces from the salvager program have member names of ZFSSLVnnn. Traces that come from the ioeFsutil program have member names that start with FSUTLnnn..nnn starts with 01 and increments for each trace output. nnn is reset to 01 when zFS is started (or restarted). See [Chapter 8, “Performance and debugging,” on page 69.](#) This is not a required parameter. If it is not specified, only a dump is generated if an abend occurs.

**Default value:** None

**Expected value:** The name of a PDSE data set.

**Example:**  user_running_hangdump=ON

**trace_table_size**

Specifies the size, in bytes, of the internal trace table. This is the size of the wrap-around trace table in the zFS address space and the salvager address space that is used for internal tracing that is always on. The trace can be sent to the trace_dsn by using the operator MODIFY ZFS,TRACE,PRINT command. You can set the trace_table_size up to 2048M, but to print the trace to a PDSE you must limit its size to 750 M.

**Default value:**
- 16 M for the zFS address space
- 64 M for the salvager address space

**Expected value:** A number in the range 1M - 2048 M.

**Example:**  trace_table_size=1M

**user_running_hangdump**

Specifies that if a user task appears to be hung for approximately 5 minutes, a dump of the user address space will be obtained by the ZFS hang detector. This dump will be with abend code 2C3 and reason code EA5805DB. This dump is accompanied by message IOEZ00605I. Use this message description to diagnose the problem.

**Default value:** Off
**Expected value:** On or Off

**Example:** user_running_hangdump=ON

**xcf_trace_table_size**

Specifies the size of the XCF trace table.

**Default value:** 4M

**Expected value:** A number in the range 1M - 2048M

**Example:** xcf_trace_table_size=8M

**Examples**

Following is a sample IOEFSPRM file that contains program options.

```
# zFS Sample Parameter File: IOEFSPRM
# For a description of these and other zFS parameters, refer to the
# zFS Administration document.
# Notes:
# 1. The IOEFSPRM file and parameters in the file are optional but it
#    is recommended that the parameter file be created in order to be
#    referenced by the DDNAME=IOEZPRM statement the PROCLIB JCL for
#    the zFS started task or through the IOEPRMxx parmlib member.
# 2. An asterisk in column 1 identifies a comment line.
# 3. A parameter specification must begin in column 1.
# The following msg_output_dsn parameter defines the optional output
# message data set. If this parameter is not specified, or if the data
# set is not found, messages will be written to the system log.
# You must delete the * from a line to activate the parameter.
# msg_output_dsn=usera.zfs.msg.out
#************************************************************************
# The following msg_input_dsn parameter is ONLY required if the optional
# NLS feature is installed. The parameter specifies the
# message input data set containing the NLS message text which is
# supplied by the NLS feature. If this parameter is not specified or if
# the data set is not found, English language messages will be generated
# by zFS. You must delete the * from a line to activate the parameter.
# msg_input_dsn=usera.sioemjpn
#************************************************************************
# The following are examples of some of the optional parameters that
# control the sizes of caches, tuning options, and program operation.
# You must delete the * from a line to activate a parameter.
#************************************************************************
+adm_threads=5
+aggrfull(90,5)
+aggrgrow=on
+change_aggrversion_on_mount=off
+client_cache_size=32M
+client_reply_storage=10M
+cmd_trace=off
+convert_auditfid=off
+convertto5=off
+file_threads=40
+format_aggrversion=4
+fsfull(88,5)
+group=IOEZFS1
+log_cache_size=32M
+meta_cache_size=64M
+metaback_cache_size=64M
```
*romount_recovery=off
*recovery_max_storage=128M
*sync_interval=45
*sysplex=filesys
*sysplex_filesys_sharemode=norwshare
*token_cache_size=65536
*tran_cache_size=4000
*user_cache_size=256M
*vnode_cache_size=131072

**********************************************************************
* The following are examples of some of the options that control zFS
* debug facilities. These parameters are not required for normal
* operation and should only be specified on the recommendation of IBM.
* You must delete the * column from a line to activate a parameter.
**********************************************************************
*debug_settings_dsn=usera.zfs.debug(file1)
*trace_dsn=usera.zfs.trace.out
*trace_table_size=1M
*xcf_trace_table_size=8M
Chapter 13. zFS application programming interfaces

This section contains programming interface information.

This topic describes the zFS application programming interface (API), `pfsctl` (BPX1PCT). Information is presented about the following ZFS commands and subcommands, which are used to manage zFS aggregates and file systems and to query and set configuration options:

- ZFSCALL_AGGR (0x40000005)
- ZFSCALL_FILESYS (0x40000004)
- ZFSCALL_CONFIG (0x40000006)
- ZFSCALL_STATS (0x40000007)

The z/OS UNIX `pfsctl` (command X'C000000B') can also retrieve zFS reason code text. For more information, see the description of the PC%ErrorText pfsctl command in the usage notes for the BPX1PCT service in z/OS UNIX System Services Programming: Assembler Callable Services Reference.

For information about how to invoke the pfsctl (BPX1PCT) application programming interface in a 64-bit environment, refer to Appendix A, “Running the zFS pfsctl APIs in 64-bit mode,” on page 327.

This topic also describes a zFS `w_pioctl` call for `fileinfo`.

### pfsctl (BPX1PCT)

#### Purpose

The pfsctl (BPX1PCT) application programming interface is used to send requests to a physical file system. It is documented in z/OS UNIX System Services Programming: Assembler Callable Services Reference. zFS is a physical file system and supports several (zFS-specific) pfsctl functions, which are documented in this section.

#### Format

```c
BPX1PCT (File_system_type, Command, Argument_Length, Argument, Return_value, Return_code, Reason_code);
```

#### Parameters

- **File_system_type**
  
  An eight-character field. In the case of zFS, it contains the characters ZFS, followed by five blanks.

- **Command**
  
  An integer. There are four major ZFS commands:
  - ZFSCALL_FILESYS (0x40000004)
Each of these commands has a set of subcommands.

**Argument Length**
An integer that contains the length of the argument.

**Argument**
A structure that has the pfsctl parameters followed by the subcommand parameters. The definitions of any structures that have padding bytes added by the compiler, have the padding bytes explicitly declared in the examples.

The following list shows the general format of the Argument for all subcommands, where \( n \) depends on the particular subcommand:

- Subcommand operation code int
- Parameter0 int
- Parameter1 int
- Parameter2 int
- Parameter3 int
- Parameter4 int
- Parameter5 int
- Parameter6 int
- Buffer\([n]\) char\([n]\)

**Return value**
An integer that contains 0 if the request is successful or -1 if it is not successful.

**Return Code**
An integer in which the return code is stored. See [z/OS UNIX System Services Messages and Codes](https://www.ibm.com/support/knowledgecenter/S5827D_1.3.2/com.ibm.zos.v1r11.doc/cd32p/32p_glossary.html) for these codes.

**Reason Code**
An integer that stores the reason code. If this code is of the form 0xEFnnxxxx, see [z/OS Distributed File Service Messages and Codes](https://www.ibm.com/support/knowledgecenter/S5827D_1.3.2/com.ibm.zos.v1r11.doc/cd32p/32p_glossary.html). Otherwise, see [z/OS UNIX System Services Messages and Codes](https://www.ibm.com/support/knowledgecenter/S5827D_1.3.2/com.ibm.zos.v1r11.doc/cd32p/32p_glossary.html).

**Usage notes**

There are four major commands, which are summarized in Table 10 on page 209 and described in detail in the following sections. The zFS pfsctl APIs will work across sysplex members. That is, zFS pfsctl APIs can query and set information on zFS aggregates owned by the current system; they can also access and set file system information from other systems in the sysplex.

The z/OS UNIX pfsctl (command X'C000000B') can also retrieve zFS reason code text. For more information, see the description of the PC#ErrorText pfsctl command in the usage notes for the BPX1PCT service in [z/OS UNIX System Services Programming: Assembler Callable Services Reference](https://www.ibm.com/support/knowledgecenter/S5827D_1.3.2/com.ibm.zos.v1r11.doc/cd32p/32p_glossary.html).
Table 10. Summary of zFS pfsctl APIs

<table>
<thead>
<tr>
<th>Command</th>
<th>Code</th>
<th>Subcommands and opcodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate</td>
<td>ZFSCALL_AGGR (0x40000005)</td>
<td>• Attach Aggregate (105)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Define Aggregate (139)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Delete File System (136)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Detach Aggregate (104)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Format Aggregate (134)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Grow Aggregate (129)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• List Aggregate Status (137)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• List Aggregate Status (Version 2) (146)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• List Attached Aggregate Names (135)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• List Attached Aggregate Names (Version 2) (140)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• List File System Names (138)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• List File System Names (Version 2) (144)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Quiesce Aggregate (132)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set Auditfid (149)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Unquiesce Aggregate (133)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Reset backup flag (157)</td>
</tr>
<tr>
<td>File System</td>
<td>ZFSCALL_FILESYS (0x40000004)</td>
<td>• List File System Status (142)</td>
</tr>
<tr>
<td>Statistics</td>
<td>ZFSCALL_STATS (0x40000007)</td>
<td>• Statistics directory cache information (249)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Statistics iobyaggr information (244)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Statistics iobydasd information (245)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Statistics icounts information (243)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Statistics kernel information (246)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Statistics locking information (240)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Statistics log cache information (247)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Statistics metadata cache information (248)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Statistics storage information (241)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Statistics transaction cache information (250)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Statistics user data cache information (242)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Statistics vnode cache information (251)</td>
</tr>
</tbody>
</table>
Table 10. Summary of zFS pfsctl APIs (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Code</th>
<th>Subcommands and opcodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>ZFSCALL_CONFIG (0x40000006)</td>
<td>• List Systems (174)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query adm_threads setting (180)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query aggrfull setting (181)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query aggrgrow setting (182)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query change_aggrversion_on_mount (246)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query client_cache_size (231)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query client_reply_storage (223)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query cmd_trace (184)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query convert_auditfid (237)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query converttov5 (250)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query debug_settings_dsn setting (186)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query file_threads (217)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query format_aggrversion (248)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query fsfull setting (187)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query group setting (214)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query log_cache_size setting (193)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query meta_cache_size setting (198)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query metaback_cache_size setting (199)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query msg_input_dsn setting (200)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query msg_output_dsn setting (201)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query sync_interval setting (205)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query syslevel (238)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query sysplex_filesys_sharemode (244)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query sysplex_state (215)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query token_cache_size (216)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query trace_dsn setting (206)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query trace_table_size setting (207)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query tran_cache_size setting (208)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query user_cache_size setting (210)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Query vnode_cache_size setting (212)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set adm_threads (150)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set aggrfull (158)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set aggrgrow (171)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set change_aggrversion_on_mount (245)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set client_cache_size (230)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set client_reply_storage (222)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set convert_auditfid (236)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set converttov5 (249)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set file_threads (176)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set format_aggrversion (247)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set fsfull (157)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set log_cache_size (153)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set meta_cache_size (152)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set metaback_cache_size (163)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set msg_output_dsn (161)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set sync_interval (154)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set sysplex_filesys_sharemode (243)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set token_cache_size (177)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set trace_dsn (159)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set tran_cache_size (160)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set user_cache_size (151)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set vnode_cache_size (155)</td>
</tr>
</tbody>
</table>

Table 11. Summary of zFS w_pioctls

<table>
<thead>
<tr>
<th>Command</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileinfo</td>
<td>0x0000A901</td>
</tr>
</tbody>
</table>
**Attach Aggregate**

**Purpose**

This subcommand call is an aggregate operation that attaches an aggregate to a system. This makes the aggregate and all its file systems known to the zFS physical file system running on that system. (Compatibility mode aggregates are attached during mount so that a separate attach is not necessary.)

**Format**

<table>
<thead>
<tr>
<th>syscall_parmlist</th>
<th>opcode</th>
<th>AGOP_ATTACH_PARMDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>parms[0]</td>
<td>offset to AGGR_ID</td>
<td></td>
</tr>
<tr>
<td>parms[1]</td>
<td>offset to AGGR_ATTACH</td>
<td></td>
</tr>
<tr>
<td>parms[2]</td>
<td>offset to system name (optional)</td>
<td></td>
</tr>
<tr>
<td>parms[3]</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>parms[4]</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>parms[5]</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>parms[6]</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**AGGR_ID**

- aid_len: char sizeof(AGGR_ID)
- aid_ver: char 1
- aid_name: char[45] "OMVS.PRV.AGGR001.LDS0001"
- aid_reserved: char[33] 0

**AGGR_ATTACH**

- at_eye: char[4] "AGAT"
- at_len: short sizeof(AGGR_ATTACH)
- at_ver: char 1
- at_res1: int 0
- at_threshold: char 90
- at_increment: char 5
- at_flags: char 0x80
- ATT_MONITOR: 0x80 Monitor aggregate full
- ATT_RDONLY: 0x80 Attach aggregate as read-only
- ATT_NBS: 0x20 Use New Block Security
- ATT_NONBS: 0x10 No longer supported
- ATT_GROW: 0x04 Allow dynamic grow
- ATT_NOGROW: 0x02 Disallow dynamic grow
- at_res2: char 0
- at_reserved: int[64] 0 reserved for future use
- systemname: char[9]

**Return_value**

0 if request is successful, -1 if it is not successful

**Return_code**

- EEXIST: Aggregate already attached
- EINTR: ZFS is shutting down
- EMSVSErr: Internal error using an osi service
- EPERM: Permission denied to perform request
- EINVAL: Attempt to attach a multi-file system aggregate

**Reason_code**

0xEFnnxxxx See z/OS Distributed File Service Messages and Codes

**Usage notes**

- The ATT_NBS and ATT_NONBS flags are no longer supported; zFS always runs with NBS on. If either of these parameters is specified, it is ignored.
- ATT_GROW and ATT_NOGROW are mutually exclusive. If neither is specified, the default is the aggrgrow setting in the IOEFSPRM file. See “Dynamically growing a compatibility mode aggregate” on page 28 for a description of dynamic grow.
- The at_threshold and at_increment values are ignored unless ATT_MONITOR is set.
- Reserved fields and undefined flags must be set to binary zeros.
Privilege required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

Detach Aggregate

Restrictions

None.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>
#define ZFSCALL_AGGR 0x40000005
#define AGOP_ATTACH_PARMDATA 105

typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;
#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t {
    char aid_eye[4]; /* Eye Catcher */
    char aid_len; /* Length of this structure */
    char aid_ver; /* Initial version */
    char aid_name[ZFS_MAX_AGGRNAME+1]; /* aggr name, null terminated */
    char aid_reserved[33]; /* Reserved for the future */
} AGGR_ID;

typedef struct aggr_attach_t {
    char at_eye[4]; /* Eye catcher */
    short at_len; /* Length of structure */
    char at_ver; /* Version */
    char at_res1; /* Reserved for internal use */
    char at_threshold; /* Threshold for monitoring */
    char at_increment; /* Increment */
    char at_flags; /* Processing flags */
    #define ATT_MONITOR 0x80 /* aggrfull monitoring should */
    /* be used */
    #define ATT_RDONLY 0x40 /* aggr should be attached ro */
    #define ATT_NBS 0x20 /* with full NBS */
    #define ATT_NOGROW 0x04 /* allow dynamic grow */
    #define ATT_NONBS 0x10 /* no longer supported */
    char at_res2; /* Reserved for future use */
    int atReserved[64]; /* Reserved for future use */
    char systemname[9];
} struct parmstruct

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char aggrname[45] = "PLEX.JMS.AGGR001.LDS0001"; /* aggregate name to attach */

    struct parmstruct myparmstruct;
```
### Define Aggregate

**Purpose**

This subcommand call is an aggregate operation that defines (creates) a VSAM linear data set, which can then be formatted as a zFS aggregate.
Define Aggregate

Format

```c
typedef struct syscall_parmlist_t {
  int opcode; /* Operation code to perform */
  int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must have sufficient authority to create the VSAM linear data set.

Related services

- Format Aggregate

Restrictions

The VSAM linear data set to be defined cannot already exist

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_DEFINE_PARM_DATA 139

typedef struct syscall_parmlist_t {
  int opcode; /* Operation code to perform */
  int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;
```

Return_value

0 if request is successful, -1 if it is not successful

Return_code

- EINT  ZFS is shutting down
- EINVAL Invalid parameters
- EMVSERR Internal error using an osi service
- ENOENT Aggregate is not attached
- EPERM Permission denied to perform request

Reason_code

- 0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
Define Aggregate

```c
#define ZFS_MAX_SMSID 8
#define ZFS_MAX_VOLVID 6
typedef struct aggr_define_t
{
    char eye[4]; /* Eye catcher */
#define ADEF_EYE "AGDF"
    short len; /* Length of this structure */
#define ADEF_VER_INITIAL 1 /* Initial version */
    char aggrName[ZFS_MAX_AGGRNAME+1];
    char dataClass[ZFS_MAX_SMSID+1];
    char managementClass[ZFS_MAX_SMSID+1];
    char storageClass[ZFS_MAX_SMSID+1];
    char model[ZFS_MAX_AGGRNAME+1];
    char modelCatalog[ZFS_MAX_AGGRNAME+1];
    char catalog[ZFS_MAX_AGGRNAME+1];
    char volumes[59][ZFS_MAX_VOLVID+1];
    char reservedChars1;
    int numVolumes;
    int spaceUnit;
#define ZFS_SPACE_CYLS 1
#define ZFS_SPACE_KILO 2
#define ZFS_SPACE_MEGA 3
#define ZFS_SPACE_RECS 4
#define ZFS_SPACE_TRKS 5
    unsigned int spacePrimary;
    unsigned int spaceSecondary;
    int reservedInts1[32];
} AGGR_DEFINE;

struct parmstruct {
    syscall_parmlist myparms;
    AGGR_DEFINE aggdef;
    char Buffer[1024];
    char systemname[9];
};

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char aggrname[45] = "PLEX.JMS.AGGR007.LDS0007"; /* aggregate name to define */
    char dataclass[9] = "";
    char managementclass[9] = "";
    char storageclass[9] = "";
    char model[45] = "";
    char modelcatalog[45] = "";
    char catalog[45] = "";
    char volumes[7] = "CFC000";

    struct parmstruct myparmstruct;
    AGGR_DEFINE *agp = &(myparmstruct.aggdef);
    char *bufp = &(myparmstruct.Buffer[0]);

    /* This next field should only be set if parms[3] is non-zero */
    /* strcpy(myparmstruct.systemname,"DCEIMGVN"); */ /* set system to run define on */
    myparmstruct.myparms.opcode = AGOP_DEFINE_PARMMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(myparmstruct.Buffer);
    myparmstruct.myparms.parms[2] = myparmstruct.myparms.parms[0]+sizeof(AGGR_DEFINE); /* offset to Buffer */
    my parmstruct.myparms.parms[3] = 0;

    /* Only specify a non-zero offset for the next field (parms[3]) if */
    /* you are running z/OS 1.7 and above, and */
    /* you want the define to run on a different system than this one */
    my parm struct.myparms.parms[3] = 0;
    my parm struct.myparms.parms[0]=sizeof(AGGR_DEFINE)+sizeof(myparmstruct.Buffer); /*
    memset(agp,0,sizeof(*agp));
    strcpy(agp->eye,ADEF_EYE); /* If included next 4 can be null */
    strcpy(agp->aggrName,aggrname);
    strcpy(agp->model,model);
    strcpy(agp->dataClass,dataclass);
    strcpy(agp->managementClass,managementclass);
    strcpy(agp->storageClass,storageclass);
    strcpy(agp->modelCatalog,modelCatalog);```

Chapter 13. zFS application programming interfaces  215
Define Aggregate

```c
strcpy(agp->volumes[0],(char *)volumes);
agp->numVolumes=1;
agp->spaceUnit=ZFS_SPACE_CYLs;
agp->spacePrimary=10;
agp->spaceSecondary=1;

BPX1PCT("ZFS",
  ZFS_CALLAgregar,
  sizeof(myparmstruct),
  (char *)&myparmstruct,
  &bpxrv,
  &bpxrc,
  &bpxrs);
if (bpxrv < 0) {
  printf("define: Error defining LDS %s\n", aggrname);
  printf("define: BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
  printf("define: Job output:\n\n%s\n",myparmstruct.Buffer);
  return bpxrc;
} else {
  printf("define: LDS %s defined successfully\n", aggrname);
}
return 0;
```
Delete File System

Format

```
syscall_parmlist
  opcode   136   AGOP_DELETEFILESYS_PARMDATA
  parms[0] offset to FS_ID or FS_ID2
  parms[1] 0
  parms[2] 0
  parms[3] 0
  parms[4] 0
  parms[5] 0
  parms[6] 0
FS_ID or FS_ID2
  fsid_eye char[4] "FSID"
  fsid_len char sizeof(FS_ID)
  fsid_ver char 1
  fsid_res1 char 0
  fsid_res2 char 0
  fsid_id hyper
    high long 0
    low long 0
  fsid_aggrname char[45] 0
  fsid_name char[45] "OMVS.PRV.FS3"
  fsid_reserved char[32] 0
  fsid_reserved2 char[2] 0
FS_ID2 or FS_ID
  fsid_eye char[4] "FSID"
  fsid_len char sizeof(FS_ID2)
  fsid_ver char 2
  fsid_res1 char 0
  fsid_res2 char 0
  fsid_id high unsigned long 0
    low unsigned long 0
  fsid_aggrname char[45] 0
  fsid_name char[45] "OMVS.PRV.FS3"
  fsid_mtname char[45] 0
  fsid_reserved char[49] 0

Return_value 0 if request is successful, -1 if it is not successful
```

Usage notes

- Removing a backup file system does not remove the associated read/write file system.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

- Attach Aggregate

Restrictions

The aggregate cannot be quiesced or attached as read-only. You cannot delete a file system that is mounted.
Delete File System

When using an FS_ID2 as input, you cannot specify the file system with the z/OS UNIX file system name (fsid_mtname) since the file system cannot be mounted. You must use the zFS file system name (fsid_name).

When FS_ID2 is used, you cannot specify the z/OS UNIX file system name (fsid_mtname).

Examples

Example 1: The following example uses an FS_ID; see Example 2 for an example that uses an FS_ID2.

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>
#define ZFSCALL_AGGR 0x40000005
#define AGOP_DELETEFILESYS_PARMDATA 136
typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    / * provides access to the parms */
    / * parms[4]-parms[6] are currently unused */
} syscall_parmlist;

typedef struct hyper { /* unsigned 64 bit integers */
    unsigned long high;
    unsigned long low;
} hyper;
#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44
typedef struct fs_id_t {
    char fsid_eye[4]; /* Eye catcher */
    #define FSID_EYE "FSID"
    char fsid_len; /* Length of this structure */
    char fsid_ver; /* Version */
    #define FSID_VER_INITIAL 1 /* Initial version */
    char fsid_res1; /* Reserved. */
    char fsid_res2; /* Reserved. */
    hyper fsid_id; /* Internal identifier */
    char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string */
    char fsid_name[ZFS_MAX_FSYSNAME+1]; /* Name, null terminated */
    char fsid_reserved[32]; /* Reserved for the future */
    char fsid_reserved2[2]; /* Reserved for the future */
} FS_ID;

struct parmstruct {
    syscall_parmlist myparms;
    FS_ID fsid;
};

int main(int argc, char **argv) {
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char filesystemname[45] = "OMVS.PRV.FS3";
    struct parmstruct myparmstruct;
    FS_ID *idp = &(myparmstruct.fsid);
    myparmstruct.myparms.opcode = AGOP_DELETEFILESYS_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = 0;
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;
    memset(idp,0,sizeof(FS_ID)); /* Ensure reserved fields are 0 */
    memcpy(&myparmstruct.fsid.fsid_eye, FSID_EYE, 4);
    myparmstruct.fsid.fsid_len = sizeof(FS_ID);
    myparmstruct.fsid.fsid_ver = FSID_VER_INITIAL;
    strcpy(myparmstruct.fsid.fsid_name, filesystemname);
    BPX1PCT("ZFS *
            ZFSCALL_AGGR, /* Aggregate operation */
```
delete file system

sizeof(myparmstruct), /* Length of Argument */
(char *) &myparmstruct, /* Pointer to Argument */
&bpxrv, /* Pointer to Return_value */
&bpxrc, /* Pointer to Return_code */
&bpxrs); /* Pointer to Reason_code */

if (bpxrv < 0)
{
    printf("Error deleting file system %s\n",filesystemname);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    return bpxrv;
}
else /* Return from delete file system was successful */
{
    printf("File system %s deleted successfully\n",filesystemname);
}
return 0;

Example 2: The following example uses an FS_ID2; see Example 2 for an example that uses an FS_ID.

#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>
define ZFSCALL_AGGR 0x40000005
#define AGOP_DELETEFILESYS_PARMDATA 136
typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;
typedef struct hyper { /* unsigned 64 bit integers */
    unsigned long high;
    unsigned long low;
} hyper;
define ZFS_MAX_AGGRNAME 44
define ZFS_MAX_FSYSNAME 44
typedef struct fs_id_t {
    char fsid_eye[4]; /* Eye catcher */
    define FSID_EYE "FSID"
    char fsid_len; /* Length of this structure */
    char fsid_ver; /* Version */
    define FSID_VER_INITIAL 1 /* Initial version */
    char fsid_res1; /* Reserved. */
    char fsid_res2; /* Reserved. */
    hyper fsid_id; /* Internal identifier */
    char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string */
    char fsid_name[ZFS_MAX_FSYSNAME+1]; /* Name, null terminated */
    char fsid_reserved[32]; /* Reserved for the future */
    char fsid_reserved2[2]; /* Reserved for the future */
} FS_ID;
typedef struct fs_id2_t {
    char fsid_eye[4]; /* Eye catcher */
    define FSID_EYE "FSID"
    char fsid_len; /* Length of this structure */
    char fsid_ver; /* Version */
    char fsid_res1; /* Reserved. */
    char fsid_res2; /* Reserved. */
    hyper fsid_id; /* Internal identifier */
    define FSID_VER_2 2 /* Second version */
    char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string */
    char fsid_name[ZFS_MAX_FSYSNAME+1]; /* Name, null terminated */
    char fsid_mntname[ZFS_MAX_FSYSNAME+1]; /* Mount name, null terminated */
    char fsid_reserved[48]; /* Reserved for the future */
} FS_ID2;
struct parmstruct
{
    syscall_parmlist myparms;
    FS_ID2 fsid;
};
int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char filesystemname[45] = "OMVS.PRV.FS3";
    struct parmstruct myparmstruct;
Delete File System

FS_ID2 *idp = &(myparmstruct.fsid);

myparmstruct.myparms.opcode = AGOP_DELETEFILESYS_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = 0;
myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(idp,0,sizeof(FS_ID2)); /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.fsid.fsid_eye, FSID_EYE, 4);
myparmstruct.fsid.fsid_len = sizeof(FS_ID2);
myparmstruct.fsid.fsid_ver = FSID_VER_2;
strcpy(myparmstruct.fsid.fsid_name,filesystemname);

BPX1PCT("ZFS ",
ZFSCALL_AGGR, /* Aggregate operation */
sizeof(myparmstruct), /* Length of Argument */
(char *) &myparmstruct, /* Pointer to Argument */
&bpxrv, /* Pointer to Return_value */
&bpxrc, /* Pointer to Return_code */
&bpxrs); /* Pointer to Reason_code */

if (bpxrv < 0)
{
    printf("Error deleting file system %s\n",filesystemname);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else /* Return from delete file system was successful */
{
    printf("File system %s deleted successfully\n",filesystemname);
    return 0;
}

Detach Aggregate

Purpose

This subcommand call is an aggregate operation that detaches an attached (but not mounted) compatibility mode aggregate. Mounted compatibility mode aggregates are detached during unmount.

Format

syscall_parmlist
opcode 104 AGOP_DETACH_PARMDATA
parms[0] offset to AGGR_ID
parms[1] 0
parms[2] 0
parms[3] 0
parms[4] 0
parms[5] 0
parms[6] 0
AGGR_ID
aid_eye char[4] "AGID"
aid_len char sizeof(AGGR_ID)
aid_ver char 1
aid_name char[45] "OMVS.PRV.AGGR001.LDS0001"
aid_reserved char[33] 0

Return_value 0 if request is successful, -1 if it is not successful

Return_code
EBUSY Aggregate could not be detached due to mounted file system
EINTR ZFS is shutting down
ENOSERR Internal error using an osi service
ENODNT Aggregate is not attached
EPERM Permission denied to perform request

Reason_code
0x17fnnxxx See z/OS Distributed File Service Messages and Codes
Usage notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

Attach Aggregate

Restrictions

All file systems in the aggregate must be unmounted before the aggregate can be detached.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
#define ZFSCALL_AGGR 0x40000005
#define AGOP_DETACH_PARMDATA 104

typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t {
    char aid_eye[4]; /* Eye catcher */
    /* Length of this structure */
    char aid_len;
    char aid_ver; /* Version */
    /* Initial version */
    char aid_name[ZFS_MAX_AGGRNAME+1]; /* Name, null terminated */
    char aid_reserved[33]; /* Reserved for the future */
} AGGR_ID;

struct parmstruct
{
    syscall_parmlist myparms;
    AGGR_ID aggr_id;
};

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";

    struct parmstruct myparmstruct;

    myparmstruct.myparms.opcode = AGOP_DETACH_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = 0;
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    memset(&myparmstruct.aggr_id,0,sizeof(AGGR_ID)); /* Ensure reserved fields are 0 */

    memcpy(&myparmstruct.aggr_id.AID_EYE,4); /* Aggregate operation */
    myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
    myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
    strcpy(myparmstruct.aggr_id.aid_name,aggrname);

    BPX1PCT("ZFS",
        ZFSCALL_AGGR, /* Aggregate operation */
        sizeof(myparmstruct), /* Length of Argument */
        (char *) &myparmstruct, /* Pointer to Argument */
    );
}
```
Detach Aggregate

```c
if (bpxrv < 0)
{
    printf("Error detaching aggregate %s\n", aggrname);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
    return bpxrc;
}
else /* Return from detach was successful */
{
    printf("Aggregate %s detached successfully\n", aggrname);
}
return 0;
```

Format Aggregate

**Purpose**

This subcommand call is an aggregate operation that formats a VSAM linear data set as a zFS aggregate. It supports both version 1.4 aggregates and version 1.5 aggregates.
Usage notes

- Reserved fields and undefined flags must be set to binary zeros.
- The af_compat bit is ignored. The VSAM LDS is always formatted as a compatibility mode aggregate.

Privilege required

The issuer must have ALTER authority on the VSAM linear data set to be formatted or must be logged in as root or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

Define Aggregate
Restrictions

The VSAM linear data set to be formatted cannot be attached.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>
#define ZFSCALL_AGGR 0x40000005
#define AGOP_FORMAT_PARMDATA 134

typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation. */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused */
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t {
    char aid_eye[4]; /* Eye catcher */
    char aid_len; /* Length of this structure */
    char aid_ver; /* Version */
    char aid_name[ZFS_MAX_AGGRNAME+1]; /* Name, null terminated */
    char aid_reserved[33]; /* Reserved for the future */
} AGGR_ID;

typedef struct aggr_format_t {
    char af_eye[4]; /* Eye catcher */
    short af_len; /* Length of structure */
    char af_ver; /* Version of cb */
    char af_aggrversion; /* 0 means honor format_aggrversion value */
    long af_size; /* Amount to format of aggr */
    long af_res1; /* For future use */
    long af_size; /* Amount to format of aggr */
    long af_logsize; /* Size of logfile in aggr */
    long af_initialempty; /* Initial empty blocks */
    int af_overwrite; /* Overwrite aggr if its not empty */
    int af_overwrite; /* Overwrite off, that means if aggr not empty it will */
    int af_compat; /* HFS-compat aggr desired */
    int af_owner; /* Owner for HFS-compat */
    int af_perms; /* Perms for HFS-compat */
    int af_perms; /* Perms for HFS-compat */
    int af_grow; /* Amount to extend each time until we reach desired size */
    int af_newauditfid; /* Old format auditfid */
    char af_reserved[56]; /* For future use */
} AGGR_FORMAT;

typedef struct parmstruct {
    syscall_parmlist myparms;
    AGGR_ID aid;
    AGGR_FORMAT aggformat;
    char systemname[9];
} myparmstruct;

int main(int argc, char **argv) {
    int bpxrv;
    /* More code here */
    return bpxrv;
}
```
int bpxrc;
int bpxrs;
char aggrname[45] = "PLEX.JMS.AGGR007.LDS0007"; /* aggregate name to format */
AGGR_FORMAT *aggptr = &(myparmstruct.aggformat);
AGGR_ID *idp = &(myparmstruct.aid);
/* This next field should only be set if parms[2] is non-zero */
/* strcpy(myparmstruct.systemname,"DCEIMGVN"); */ /* set system to change */
/* Only specify a non-zero offset for the next field (parms[2]) if */
/* you are running z/OS 1.7 and above, and */
/* you want the format to be run on a different system than this one */
/* myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist)+sizeof(AGGR_ID)+sizeof(AGGR_FORMAT); */
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;
memset(idp,0,sizeof(AGGR_ID));
memcpy(idp->aid_eye,AID_EYE,4);
idp->aid_ver=1;
strcpy(idp->aid_name,aggrname);
idp->aid_len=(int) sizeof(AGGR_ID);
memset(aggptr,0,sizeof(myparmstruct.aggformat));
memcpy(aggptr->af_eye,AF_EYE,4);
aggptr->af_len = sizeof(myparmstruct.aggformat);
aggptr->af_size = AF_DEFAULT_SIZE;
aggptr->af_compat = AF_HFSCOMP; /* I want an HFS compatibility mode aggregate */
aggptr->af_ownerSpecified = AF_OWNER_USECALLER; /* aggptr->af_owner = owner; */
aggptr->af_groupSpecified = AF_GROUP_USECALLER; /* aggptr->af_group = group; */
aggptr->af_permsSpecified = AF_PERMS_DEFAULT; /* aggptr->af_perms = perms; */
aggptr->af_grow = 0; /* no grow size */
aggptr->af_aggrversion = 0; /* format with default version defined by format_aggrversion value */
aggptr->af_newauditfid = 1; /* generate a new auditfid */
(zfSCALL_AGGR, /* Aggregate operation */
 sizeof(myparmstruct), /* Length of Argument */
 (char *) &myparmstruct, /* Pointer to Argument */
 &bpxrv, /* Pointer to Return_value */
 &bpxrc, /* Pointer to Return_code */
 &bpxrs); /* Pointer to Reason_code */
if (bpxrv < 0) {
    printf("Error formatting, BPRV = %d\n BPXRC = %d\n BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else {
    printf("Formatted aggregate %s\n",aggrname);
}
return 0;

**Grow Aggregate**

**Purpose**

This subcommand extends the physical size of an attached aggregate. It supports both version 1.4 aggregates and version 1.5 aggregates.
Grow Aggregate

Format

ssyscall_parmlist
opcode 129 AGOP_GROW_PARMDATA
parms[0] offset to AGGR_ID
parms[1] new size of aggregate
parms[2] 0
parms[3] 0
parms[4] 0
parms[5] 0
parms[6] 0
AGGR_ID
aid_eye char[4] "AGID"
aid_len char sizeof(AGGR_ID)
aid_ver char 1 /* new size is 32 bits */
aid_name char[45] "OMVS.PRV.AGGR001.LDS0001"
aid_reserved char[33] 0 /* Reserved for the future */

- OR -

ssyscall_parmlist
opcode 129 AGOP_GROW_PARMDATA
parms[0] offset to AGGR_ID
parms[1] high 32 bits of new 64 bit size of aggregate
parms[2] low 32 bits of new 64 bit size of aggregate
parms[3] 0
parms[4] 0
parms[5] 0
parms[6] 0
AGGR_ID
aid_eye char[4] "AGID"
aid_len char sizeof(AGGR_ID)
aid_ver char 3 /* new size is 64 bits */
/* Note: version 2 is only used with */
/* List Attached Aggregate Names (Version 2) */
aid_name char[45] "OMVS.PRV.AGGR001.LDS0001"
aid_reserved char[33] 0 /* Reserved for the future */

Return_value 0 if request is successful, -1 if it is not successful

Return_code
8 DFSMS did not extend the aggregate
EBUSY Aggregate is busy or otherwise unavailable
EINVAL Invalid parameters
EMVSERR Internal error using an osi service
ENOENT No aggregate by this name is found
EPERM Permission denied to perform request

Reason_code
0xEFnnxxxx See z/OS Distributed File Service Messages and Codes

Usage notes

- The aggregate must be mounted or attached.
- The size specified is the new total size (in 1 KB blocks) that is being requested. The size can be rounded up by DFSMS. If a zero is specified for the new size, the aggregate is grown by a secondary allocation. DFSMS determines whether to extend to another volume. Requests that write to files and need aggregate blocks that are not available yet and other requests that access those files will wait. Other requests will not wait during the grow.
- For an AGGR_ID version 1, the new size cannot be larger than approximately 4 TB. For an AGGR_ID version 3, the new size cannot be larger than approximately 16 TB.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must have ALTER authority on the VSAM linear data set to be formatted and must be logged in as root or have READ authority to the
SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

List Aggregate Status Version 2

Restrictions

The aggregate to be grown cannot already be quiesced or be attached as read-only. An aggregate cannot be made smaller.

Examples

• Example 1:

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
/* #include <stdlib.h> */
#define ZFSCALL_AGGR 0x40000005
#define AGOP_GROW_PARMDATA 129

typedef struct syscall_parmlist_t {
    int opcode;  /* Operation code to perform */
    int parms[7];  /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused */
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t {
    char aid_eye[4];  /* Eye catcher */
    /* #define AID_EYE "AGID" */
    char aid_len;  /* Length of this structure */
    char aid_ver;  /* Version */
    /* #define AID_VER_INITIAL 1 */
    /* Initial version */
    char aid_name[ZFS_MAX_AGGRNAME+1];  /* Name, null terminated */
    char aid_reserved[33];  /* Reserved for the future */
} AGGR_ID;

struct parmstruct {
    syscall_parmlist myparms;
    AGGR_ID aggr_id;
} ;

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";

    memset(&myparmstruct.aggr_id,0,sizeof(AGGR_ID));  /* Ensure reserved fields are 0 */
    myparmstruct.myparms.opcode = AGOP_GROW_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = 70000;  /* New size of aggregate in K-bytes */
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    memcpy(&myparmstruct.aggr_id.aid_eye,AID_EYE,4);
    myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
    myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
    strcpy(myparmstruct.aggr_id.aid_name,aggrname);

    BPX1PCT("ZFS ",
        ZFSCALL_AGGR,  /* Aggregate operation */
        sizeof(myparmstruct), /* Length of Argument */
        (char *)&myparmstruct, /* Pointer to Argument */
        (char *)&bpxrv,  /* Pointer to Return_value */
        (char *)&bpxrc,  /* Pointer to Return_code */
        (char *)&bpxrs);  /* Pointer to Reason_code */

    if (bpxrv < 0)
    {
        printf("Error growing aggregate %s\n", aggrname);
        print("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    }
}
```

Chapter 13. zFS application programming interfaces  227
Grow Aggregate

```c
else /* Return from grow was successful */
{
    printf("Aggregate %s grown successfully\n",aggrname);
    return 0;
}
```

List Aggregate Status

Purpose

This subcommand call is an aggregate operation that returns information about a specified attached aggregate on this system.

Format

```plaintext
syscall_parmlist
    opcode 137   AGOP_GETSTATUS_PARMDATA
    parms[0] offset to AGGR_ID
    parms[1] offset to AGGR_STATUS
    parms[2] 0
    parms[3] 0
    parms[4] 0
    parms[5] 0
    parms[6] 0

AGGR_ID
    aid_eye char[4] "AGID"
    aid_len char sizeof(AGGR_ID)
    aid_ver char 1
    aid_name char[45] "OMVS.PRV.AGGR001.LDS0001"
    aid_reserved char[33] 0

AGGR_STATUS
    as_eye char[4] "AGST"
    as_len short sizeof(AGGR_STATUS)
    as_ver char 1
    as_aggrId long Aggregate ID
    as_nFileSystems long Number of File Systems
    as_threshold char Aggrfull threshold
    as_increment char Aggrfull increment
    as_flags char
        AS_MONITOR 0x80
        AS_RO     0x40
        AS_NBS   0x20
        AS_COMPAT 0x10
        AS_GROW 0x08
    as_res2 char 0
    as_blocks unsigned long
    as_fragSize long
    as_blockSize long
    as_totalUsable unsigned long
    as_realFree unsigned long
    as_minFree unsigned long
    as_reserved char[128]

Return_value  0 if request is successful, -1 if it is not successful

Return_code
    EINTR     ZFS is shutting down
    EINVAL    Invalid parameter list
    EMVSERR   Internal error using an osi service
    ENOENT    Aggregate is not attached

Reason_code
    0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
```

Usage notes

- To grow an aggregate, you need to specify a number larger than the sum of `as_totalUsable` and `as_minFree`.
- Reserved fields and undefined flags must be set to binary zeros.
Privilege required

None.

Related services

List Attached Aggregate Names

Restrictions

None.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>
#define ZFSCALL_AGGR 0x40000005
#define AGOP_GETSTATUS_PARM_DATA 137

typedef struct syscall_parmlist_t {
  int opcode;  /* Operation code to perform */
  int parms[7]; /* Specific to type of operation, */
           /* provides access to the parms */
           /* parms[4]-parms[6] are currently unused */
} syscall_parmlist;
#define ZFS_MAX_AGGNAME 44

typedef struct aggr_id_t {
  char aid_eye[4];  /* Eye Catcher */
#define AID_EYE "AGID"
  char aid_len;  /* Length of this structure */
#define AID_LEN INITIAL 1
  char aid_ver;  /* Version */
#define AID_VER INITIAL 1
  char aid_name[ZFS_MAX_AGGNAME+1]; /* aggr name, null terminated */
  char aid_reserved[33]; /* Reserved for the future */
} AGGR_ID;

typedef unsigned long u_long;

typedef struct aggr_status_t {
  char as_eye[4];  /* Eye catcher */
#define AS_EYE "AGST"
  short as_len;  /* Length of structure */
#define AS_VER INITIAL 1
  char as_ver;
#define AS_VER INITIAL 1
  long as_agrid;  /* Internal identifier */
  long as_nFileSystems;  /* Number of filesystems in aggregate */
  char as_threshold;  /* Threshold for aggrfull monitoring */
  char as_increment;  /* Increment for aggrfull monitoring */
  char as_flags;
#define AS_MONITOR 0x80 /* Aggr monitored for aggr full */
#define AS_RO 0x40 /* Aggr attached Read-only */
#define AS_NBS 0x20 /* Aggr should guarantee NBS */
#define AS_COMPAT 0x10 /* Aggr is HFS compatible */
#define AS_GROW 0x08 /* Aggr can be dynamically grown */
  char as_res1;  /* Reserved */
  u_long as_blocks;  /* Number of fragments in aggregate */
#define AS_BLOCKS_SIZE INITIAL 1
  long as_fragSize;  /* Size of fragment in aggregate (normally 1K) */
  long as_blockSize;  /* Size of blocks on aggregate (normally 8K) */
  u_long as_totalUsable;  /* Total available blocks on aggregate (normally 8K) */
  u_long as_realFree;  /* Total kilobytes free */
  u_long as_minFree;  /* Minimum kilobytes free */
  char as_reserved[128];  /* Reserved for future */
} AGGR_STATUS;

struct parmstruct {
  syscall_parmlist myparms;
  AGGR_ID aggr_id;
  AGGR_STATUS aggr_status;
} ;

int main(int argc, char **argv)
{
  int bpxrv;
  int bpxrc;
  int bpxrs;
  char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";  /* aggregate name to getstatus */

  struct parmstruct myparmstruct;
```
List Aggregate Status

### List Aggregate Status (Version 2)

#### Purpose

This subcommand returns information about a specified attached aggregate on this system. Version 2 returns additional flags and fields.
### List Aggregate Status (Version 2)

#### Format

<table>
<thead>
<tr>
<th>syscall_parmlist</th>
<th>opcode</th>
<th>AGOP_GETSTATUS2_PARMDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>param[0]</td>
<td>offset</td>
<td>AGGR_ID</td>
</tr>
<tr>
<td>param[1]</td>
<td>offset</td>
<td>AGGR_STATUS2</td>
</tr>
<tr>
<td>param[2]</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>param[3]</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>param[4]</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>param[5]</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>param[6]</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

#### AGGR_ID

- **aid_eye** char[4] "AGID"
- **aid_len** char sizeof(AGGR_ID)
- **aid_ver** char 1
- **aid_name** char[45] "OMVS.PRV.AGGR001.LDS0001"
- **aid_reserved** char[33] 0

#### AGGR_STATUS2

- **as_eye** char[4] "AGST"
- **as_len** short sizeof(AGGR_STATUS2)
- **as_ver** char 2
- **as_aggrId** long Aggregate ID
- **as_nFileSystems** long Number of File Systems
- **as_threshold** char Aggrfull threshold
- **as_increment** char Aggrfull increment
- **as_flags** char
  - AS_MONITOR 0x80
  - AS_RO 0x40
  - AS_NBS 0x20
  - AS_COMPAT 0x10
  - AS_GROW 0x08
  - AS_QUIESCED 0x01
- **as_flags2** char
  - AS_DISABLED 0x80
  - AS_SYSLEXWARE 0x40
- **as_blocks** unsigned long
- **as_fragSize** long
- **as_blockSize** long
- **as_totalUsable** unsigned long
- **as_realFree** unsigned long
- **as_minFree** unsigned long
- **as_reserved2** int[3] /* reserved */
- **as_freeblocks** unsigned long
- **as_freefrags** unsigned long
- **as_directLog** unsigned long
- **as_indirectLog** unsigned long
- **as_fstbl** unsigned long
- **as_bitmap** unsigned long
- **as_diskFormatMajorVersion** unsigned long
- **as_diskFormatMinorVersion** unsigned long
- **as_auditfid** char[10]
- **as_bytes_reserved** char[2]
- **as_reserved3** long
- **as_quiesce_time** struct timeval
- **posix_time_low** int
- **posix_usecs** int
- **as_quiesce_ipname** char[9]
- **as_quiesce_sysname** char[9]
- **as_reserved** char[42]

OR (continued...)
List Aggregate Status (Version 2)

- The aggregate must be mounted or attached.
- To grow an aggregate, you need to specify a number larger than the sum of as_totalUsable and as_minFree.
- For an AGGR_STATUS2, if a size is too large for 32 bits, 0xFFFFFFFF is returned.
  For an AGGR_STATUS3, sizes are returned in both the normal fields and the hyper fields.
List Aggregate Status (Version 2)

- Reserved fields and undefined flags must be set to binary zeros.

**Privilege required**

None.

**Related services**

List Attached Aggregate Names

**Restrictions**

None.

**Examples**

**Example 1:**

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>
#define ZFSCALL_AGGR 0x00000005
#define AGOP_GETSTATUS2_PARMDATA 146

typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct timeval {
    long posix_time_low; /* seconds since epoch */
    long posix_usec; /* microseconds */
} TIMEVAL;
#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t {
    char aid_eye[4]; /* Eye Catcher */
#define AID_EYE "AGID"
    char aid_len; /* Length of this structure */
    char aid_ver; /* Version */
#define AID_VER_INITIAL 1 /* Initial version */
    char aid_name[ZFS_MAX_AGGRNAME+1]; /* aggr name, null terminated */
    char aid_reserved[33]; /* Reserved for the future */
} AGGR_ID;

typedef unsigned long u_long;

typedef struct aggr_status_t {
    char as_eye[4]; /* Eye catcher */
#define AS_EYE "AGST"
    short as_len; /* Length of structure */
    char as_ver; /* Version */
#define AS_VER_2 2 /* version 2 */
    char as_res1; /* Reserved */
    long as_aggrId; /* Internal identifier */
    long as_nFileSystems; /* Number of filesystems in aggregate */
    long as_threshold; /* Threshold for aggrfull monitoring */
    long as_increment; /* Increment for aggrfull monitoring */
    char as_flags; /* Aggregate flags */
#define AS_MONITOR 0x80 /* Aggr monitored for aggr full */
#define AS_RO 0x40 /* Aggr attached Read-only */
#define AS_NBS 0x20 /* Aggr should guarantee NBS */
#define AS_COMPAT 0x10 /* Aggr is HFS compatible */
#define AS_GROW 0x08 /* Aggr can be dynamically grown */
/* The following flags are for AS_VER_2 */
#define AS_QUIESCED 0x01 /* 1 = Aggr is quiesced, 0 = Aggr is unquiesced */
#define AS_FLAGS2 0x0c /* Aggr is sysplex-aware for r/w */
#define AS_DISABLED 0x80 /* 1 = Aggr is disabled */
#define AS_SYSPLEXAWARE 0x40 /* Aggr is sysplex-aware for r/w */
    u_long as_flags2; /* Aggregate flags2 */
    u_long as_blocks; /* Number of fragments in aggregate */
    long as_fragSize; /* Size of fragment in aggregate (normally 1K) */
    long as_blockSize; /* Size of blocks on aggregate (normally 8K) */
    u_long as_totalsize; /* Total available blocks on aggregate (normally 8K) */
    u_long as_realFree; /* Total kilobytes free */
} AGGR_ID;
```

Chapter 13. zFS application programming interfaces 233
List Aggregate Status (Version 2)

u_long as_minFree; /* Minimum kilobytes free */
int as_reserved[3];
u_long as_freeblocks; /*Number of k available in free 8k blocks*/
u_long as_freefrags; /*Number of k available in free 1k fragments*/
u_long as_directLog; /*Number of k used on the log*/
u_long as_indirectLog; /*Number of k used indirectly on the log*/
u_long as_fstbl; /*Number of k used for the filesystem table*/
u_long as_diskFormatMajorVersion; /* disk format major version */
u_long as_diskFormatMinorVersion; /* disk format minor version */
char as_auditfid[30]; /* 6 byte volser followed by 4 byte CCHH */
char as_bytes_reserved[2]; /* reserved */
long as_reserved3;
struct timeval as_quiesce_time; /* time of last quiesce */
char as_quiesce_jbname[9]; /* job name of last quiesce - null terminated */
char as_quiesce_sysname[9]; /* system where last quiesce issued - null terminated */
char as_reserved[42]; /* Reserved for future */

} AGGR_STATUS2;

struct parmstruct{
    syscall_parmlist myparms;
    AGGR_ID aggr_id;
    AGGR_STATUS2 aggr_status;
} ;

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    int i;
    char buff[33];
    char aggrname[45] = "PLEX.JMS.AGGR001.LDS0001"; /* aggregate name to getstatus */

    struct parmstruct myparmstruct;
    AGGR_ID *idp = &(myparmstruct.aggr_id);
    AGGR_STATUS2 *asp = &(myparmstruct.aggr_status);

    myparmstruct.myparms.opcode = AGGR_GETSTATUS2_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;
    memset(idp,0,sizeof(AGGR_ID)); /* Ensure reserved fields are 0 */
    memset(asp,0,sizeof(AGGR_STATUS2)); /* Ensure reserved fields are 0 */
    memcpy(&myparmstruct.aggr_status.as_eye[0], AS_EYE, 4);
    myparmstruct.aggr_status.as_len = sizeof(AGGR_STATUS2);
    myparmstruct.aggr_status.as_ver = AS_VER_2;
    memcpy(&myparmstruct.aggr_status.as_len, AS_EYE, 4);
    myparmstruct.aggr_status.as_len = sizeof(AGGR_STATUS2);
    myparmstruct.aggr_status.as_ver = AS_VER_2;

    if (bpxrv < 0)
    {
        printf("Error getstatus aggregate %s\n", aggrname);
        printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    }
    else /* Return from getstatus was successful */
    {
        printf("Aggregate %s getstatus successful\n",aggrname);
        printf("Getstatus: aggr_id=%d, no_of_filesystems=%d, aggr_flags=%2.2x, aggr_flags2=%2.2x\n", myparmstruct.aggr_status.as_aggrId, myparmstruct.aggr_status.as_nFileSystems, myparmstruct.aggr_status.as_aggrFlags, myparmstruct.aggr_status.as_aggrFlags2);
        printf("Getstatus: threshold=%d, increment=%d\n", myparmstruct.aggr_status.as_threshold, myparmstruct.aggr_status.as_increment);
        printf("Getstatus: blocks=%d, frag_size=%d, block_size=%d\n", myparmstruct.aggr_status.as_blocks, myparmstruct.aggr_status.as_fragSize, myparmstruct.aggr_status.as_blockSize);
    }
List Aggregate Status (Version 2)

printf("getstatus: total_usable=%d, real_free=%d, min_free=%d\n",
    myparmstruct.aggr_status.as_totalUsable,
    myparmstruct.aggr_status.as_realFree,
    myparmstruct.aggr_status.as_minFree);
printf("getstatus: free_8K_blocks=%d, free_1K_fragments=%d\n",
    myparmstruct.aggr_status.as_freeblocks/8,
    myparmstruct.aggr_status.as_freefrags);
printf("getstatus: direct_Log=%d, indirect_Log=%d\n",
    myparmstruct.aggr_status.as_directLog,
    myparmstruct.aggr_status.as_indirectLog);
printf("getstatus: filesystem_table=%d, bitmap=%d\n",
    myparmstruct.aggr_status.as_fstbl,
    myparmstruct.aggr_status.as_bitmap);
printf("getstatus: version=%d.%d\n",
    myparmstruct.aggr_status.as_diskFormatMajorVersion,
    myparmstruct.aggr_status.as_diskFormatMinorVersion);
printf("getstatus: auditfid=");
for (i=0; i<10; i++)
    { printf("%2.2X",myparmstruct.aggr_status.as_auditfid[i]); }
printf("\n");

if (myparmstruct.aggr_status.as_flags & AS_QUIESCED)
    {
        if (myparmstruct.aggr_status.as_quiesce_jbname[0]!=0x00)
            {
                if (0==ctime_r((time_t *)&myparmstruct.aggr_status.as_quiesce_time.posix_time_low, buf))
                    { printf("Could not get timestamp.\n"); }
                else
                    { /* Insert the microseconds into the displayable time value */
                        strncpy(&buf[22], &buf[20], 6);
                        sprintf(&buf[20], "%06d", myparmstruct.aggr_status.as_quiesce_time.posix_usecs);
                        buf[20][6] = 0;
                        printf("Quiesced by job %s on system %s on %s",
                            myparmstruct.aggr_status.as_quiesce_jbname,
                            myparmstruct.aggr_status.as_quiesce_sysname,
                            buf);
                    }
                printf("\n");
            }
        return 0;
    }

Example 2:

#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>
define ZFSCALL_AGGR 0x40000005
define AGOP_GETSTATUS2_PARMDATA 146
typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct timeval {
    long posix_time_low; /* seconds since epoch */
    long posix_usecs; /* microseconds */
} TIMEVAL;

typedef struct hyper_t { /* unsigned 64 bit integers */
    unsigned long high;
    unsigned long low;
    /* reserved for the future */
} hyper;
define ZFS_MAX_AGGRNAME 44
typedef struct aggr_id_t {
    char aid_eye[4]; /* Eye Catcher */
    define AID_EYE "AGID"
    char aid_len; /* Length of this structure */
    char aid_ver; /* Version */
    define AID_VER_INITIAL 1 /* Initial version */
    char aid_len[ZFS_MAX_AGGRNAME+1]; /* aggr name, null terminated */
    char aid_reserved[ZFS_MAX_AGGRNAME+1]; /* Reserved for the future */
    } hyper;
define ZFS_MAX_AGGRNAME 44

List Aggregate Status (Version 2)

```c
typedef struct aggr_status_t {
    char as_eye[4]; /* Eye catcher */
    short as_len;   /* Length of structure */
    char as_ver;    /* Version number */
    #define AS_VER_3 3 /* version 3 */
    short as_agrid; /* Internal identifier */
    long as_nfilesystems; /* Number of filesystems in aggregate */
    long as_nfs_instances; /* Number of NFS instances */
    char as_threshold; /* Threshold for aggrfull monitoring */
    char as_increment; /* Increment for aggrfull monitoring */
    char as_flags;    /* Aggregate flags */
    #define AS_MONITOR 0x80 /* Aggr monitored for aggrfull */
    #define AS_RO 0x40  /* Aggr attached Read-only */
    #define AS_NBS 0x20 /* Aggr should guarantee NBS */
    #define AS_COMPAT 0x10 /* Aggr is HFS compatible */
    #define AS_GROW 0x08 /* Aggr can be dynamically grown */
    char as_flags2;   /* Aggregate flags2 */
    long as_aggrId;  /* Internal identifier */
    long as_nFileSystems; /* Number of filesystems in aggregate */
    char as_threshold; /* Threshold for aggrfull monitoring */
    char as_increment; /* Increment for aggrfull monitoring */
    char as_flags;    /* Aggregate flags */
    #define AS_MONITOR 0x80 /* Aggr monitored for aggrfull */
    #define AS_RO 0x40  /* Aggr attached Read-only */
    #define AS_NBS 0x20 /* Aggr should guarantee NBS */
    #define AS_COMPAT 0x10 /* Aggr is HFS compatible */
    #define AS_GROW 0x08 /* Aggr can be dynamically grown */
    char as_flags2;   /* Aggregate flags2 */
    long as_aggrId;  /* Internal identifier */
    long as_nFileSystems; /* Number of filesystems in aggregate */
    char as_threshold; /* Threshold for aggrfull monitoring */
    char as_increment; /* Increment for aggrfull monitoring */
    char as_flags;    /* Aggregate flags */
    #define AS_MONITOR 0x80 /* Aggr monitored for aggrfull */
    #define AS_RO 0x40  /* Aggr attached Read-only */
    #define AS_NBS 0x20 /* Aggr should guarantee NBS */
    #define AS_COMPAT 0x10 /* Aggr is HFS compatible */
    #define AS_GROW 0x08 /* Aggr can be dynamically grown */
    char as_flags2;   /* Aggregate flags2 */
    long as_aggrId;  /* Internal identifier */
    long as_nFileSystems; /* Number of filesystems in aggregate */
    char as_threshold; /* Threshold for aggrfull monitoring */
    char as_increment; /* Increment for aggrfull monitoring */
    char as_flags;    /* Aggregate flags */
    #define AS_MONITOR 0x80 /* Aggr monitored for aggrfull */
    #define AS_RO 0x40  /* Aggr attached Read-only */
    #define AS_NBS 0x20 /* Aggr should guarantee NBS */
    #define AS_COMPAT 0x10 /* Aggr is HFS compatible */
    #define AS_GROW 0x08 /* Aggr can be dynamically grown */
    char as_flags2;   /* Aggregate flags2 */
} AGGR_STATUS3;
```
List Aggregate Status (Version 2)

memset(idp,0,sizeof(AGGR_ID)); /* Ensure reserved fields are 0 */
memset(asp,0,sizeof(AGGR_STATUS3)); /* Ensure reserved fields are 0 */
memcpy(&myparmstruct.aggr_status.as_eye[0], AS_EYE, 4);
myparmstruct.aggr_status.as_len = sizeof(AGGR_STATUS3);
myparmstruct.aggr_status.as_ver = AS_VER_3;
memcpy(&myparmstruct.aggr_id,AID_EYE,4);
myparmstruct.aggr_id.aid_len = sizeof(AID_ID);
myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
strcpy(myparmstruct.aggr_id.aid_name,aggrname);

BPX1PCT("ZFS
", /* Aggregate operation */
ZFSCALL_AGGR, /* Length of Argument */
(char *) &myparmstruct, /* Pointer to Argument */
&bpxrv, /* Pointer to Return_value */
&bpxrc, /* Pointer to Return_code */
&bpxrs);

if (bpxrv < 0)
{
    printf("Error getstatus aggregate %s\n", aggrname);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
    return bpxrc;
} else /* Return from getstatus was successful */
{
    printf("Aggregate %s getstatus successful\n", aggrname);
    printf("getstatus: aggr_id=%d, no_of_filesystems=%d, aggr_flags=%2.2x, aggr_flags2=%2.2x\n",
            myparmstruct.aggr_status.as_aggrId,
            myparmstruct.aggr_status.as_nFileSystems,
            myparmstruct.aggr_status.as_flags,
            myparmstruct.aggr_status.as_flags2);
    printf("getstatus: threshold=%d, increment=%d\n",
            myparmstruct.aggr_status.as_threshold,
            myparmstruct.aggr_status.as_increment);
    printf("getstatus: blocks=%d, frag_size=%d, block_size=%d\n",
            myparmstruct.aggr_status.as_blocks,
            myparmstruct.aggr_status.as_fragSize,
            myparmstruct.aggr_status.as_blockSize);
    printf("getstatus: total_usable=%d, real_free=%d, min_free=%d\n",
            myparmstruct.aggr_status.as_totalUsable,
            myparmstruct.aggr_status.as_realFree,
            myparmstruct.aggr_status.as_minFree);
    printf("getstatus: free_8K_blocks=%d, free_1K_fragments=%d\n",
            myparmstruct.aggr_status.as_freeblocks/8,
            myparmstruct.aggr_status.as_freefrags);
    printf("getstatus: direct_Log=%d, indirect_Log=%d\n",
            myparmstruct.aggr_status.as_directLog,
            myparmstruct.aggr_status.as_indirectLog);
    printf("getstatus: filesystem_table=%d, bitmap=%d\n",
            myparmstruct.aggr_status.as_fstbl,
            myparmstruct.aggr_status.as_bitmap);
    printf("getstatus: total_usableh=%d, total_usablel=%d, real_freeh=%d, real_freel=%d, min_freeh=%d, min_freel=%d\n",
            myparmstruct.aggr_status.as_totalUsable_hyper.high,
            myparmstruct.aggr_status.as_totalUsable_hyper.low,
            myparmstruct.aggr_status.as_realFree_hyper.high,
            myparmstruct.aggr_status.as_realFree_hyper.low,
            myparmstruct.aggr_status.as_minFree_hyper.high,
            myparmstruct.aggr_status.as_minFree_hyper.low);
    printf("getstatus: free_8K_blocksh=%d, free_8K_blocksl=%d, free_1K_fragmentsh=%d, free_1K_fragmentsl=%d\n",
            myparmstruct.aggr_status.as_freeblocks_hyper.high/8,
            myparmstruct.aggr_status.as_freeblocks_hyper.low/8,
            myparmstruct.aggr_status.as_freefrags_hyper.high,
            myparmstruct.aggr_status.as_freefrags_hyper.low/8,
    );
List Attached Aggregate Names

Purpose

This subcommand call is an aggregate operation that returns a list of the names of all attached aggregates on a system.
List Attached Aggregate Names

Format

```c
typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, provides access to the parms */
    int parms[4]-parms[6] are currently unused*/

    AGGR_ID[2] Array of AGGR_IDs (n can be 0)
    aid_eye char[4] "AGID"
    aid_len char sizeof(AGGR_ID)
    aid_ver char 1
    aid_name char[45] "OMVS.PRV.AGGR001.LDS0001"
    aid_reserved char[33] 0
    size needed long 0
    systemname char[9]
}
```

Usage notes

- This call returns an array of AGGR_IDs, one for each attached aggregate on the system. Each AGGR_ID structure is 84 bytes. You can specify a buffer that you think might hold all of them or you can specify a buffer length and offset of zero. If you get a return code of E2BIG, the required size for the buffer is contained in the size field.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

List Aggregate Status
List File System Names

Restrictions

None.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#define ZFS_MAX_AGGRNAME 44
```
typedef struct aggr_id_t {
    char aid_eye[4]; /* Eye Catcher */ #define AID_EYE "AGID"
    char aid_len; /* Length of this structure */
    char aid_ver; /* Version */
    #define AID_VER_INITIAL 1 /* Initial version */
    char aid_name[ZFS_MAX_AGGRNAME+1]; /* aggr name, null terminated */
    char aid_reserved[33]; /* Reserved for the future */
} AGGR_ID;

struct parmstruct
{
    syscall_parmlist myparms; /* Real malloc'd structure will have an array of AGGR_IDs here */
    long size;
    char systemname[9];
};

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;

    struct parmstruct myparmstruct;
    AGGR_ID *aggPtr;
    int aggSize = sizeof(AGGR_ID);
    int buflen = sizeof(AGGR_ID);
    struct parmstruct *myp = &myparmstruct;
    int mypsize;
    char *systemp;
    int count_aggrs, total_aggrs;

    myparmstruct.myparms.opcode = AGOP_LISTAGGRNAMES_PARMDATA;
    myparmstruct.myparms.parms[0] = 0;
    myparmstruct.myparms.parms[1] = 0;
    myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    BPX1PCT("ZFS",
        ZFSCALL_AGGR, /* Aggregate operation */
        sizeof(myparmstruct), /* Length of Argument */
        (char *) &myparmstruct, /* Pointer to Argument */
        &bpxrv, /* Pointer to Return_value */
        &bpxrc, /* Pointer to Return_code */
        &bpxrs); /* Pointer to Reason_code */

    if (bpxrv < 0)
    {
        if (bpxrc == E2BIG)
        {
            buflen = myp->size; /* Get buffer size needed */
            mypsize = buflen + sizeof(syscall_parmlist) + sizeof(long) + 9;
            myp = (struct parmstruct *) malloc ((long) mypsize);
            memset(myp, 0, mypsize);

            /* This next field should only be set if parms[3] is non-zero */
            /* set system to get lsaggr info from */
            myparmstruct.myparms.opcode = AGOP_LISTAGGRNAMES_PARMDATA;
            myparmstruct.myparms.parms[0] = buflen;
            myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist);
            myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + buflen;
            myparmstruct.myparms.parms[3] = 0;

            /* Only specify a non-zero offset for the next field (parms[3]) if */
            /* you are running z/OS 1.7 and above, and */
            /* you want lsaggr aggregates owned on a single system */
            myparmstruct.myparms.parms[3] = sizeof(syscall_parmlist) + buflen + sizeof(long);

            myparmstruct.myparms.parms[4] = 0;
            myparmstruct.myparms.parms[5] = 0;
            myparmstruct.myparms.parms[6] = 0;

            BPX1PCT("ZFS",
                ZFSCALL_AGGR, /* Aggregate operation */
                mypsize, /* Length of Argument */
                (char *) myp, /* Pointer to Argument */
                &bpxrv, /* Pointer to Return_value */
                &bpxrc, /* Pointer to Return_code */
                &bpxrs); /* Pointer to Reason_code */

            if (bpxrv == 0)
            {
                total_aggrs = buflen/aggSize;
                count_aggrs = 1;
                for(aggPtr = (AGGR_ID *) &myp->size) ;
                count_aggrs <= total_aggrs ;
            }
        }
    }
List Attached Aggregate Names

This subcommand returns a list of the names of all attached aggregates on a system with the system name.

**Format**

```
syscall_parmlist
  opcode       140   AGDP_LISTAGGRNAMES2_PARMDATA
  parms[0]     buffer length or 0
  parms[1]     offset to AGGR_ID2 or 0
  parms[2]     offset to size
  parms[3]     offset to system name (optional)
  parms[4]     0
  parms[5]     0
  parms[6]     0
  AGGR_ID[n]   Array of AGGR_ID2s (n can be 0)
  aid_eye      char[4]   "AGID"
  aid_len      char   sizeof(AGGR_ID)
  aid_ver      char   2
  aid_name     char[45]   "OMVS.PRV.AGGR001.LDS0001"
  aid_sysname  char[9]   "DCEIMGVN"
  aid_reserved char[24]   0
  size needed  long   0
  systemname   char[9]
Return_value  0 if request is successful, -1 if it is not successful
Return_code
  EINTR       ZFS is shutting down
  EINVAL      Invalid parameter list
  EMVSERR     Internal error using an osi service
  ENDFIND     Aggregate is not attached
  E2BIG       List is too big for buffer supplied
Reason_code
  0xEFnnxxxx  See z/OS Distributed File Service Messages and Codes
```
List Attached Aggregate Names (Version 2)

Usage notes

- This call returns an array of AGGR_ID2s, one for each attached aggregate on the system. Each AGGR_ID structure is 84 bytes. You can specify a buffer that you think might hold all of them or you can specify a buffer length and offset of zero. If you get a return code of E2BIG, the required size for the buffer is contained in the size field.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

List Aggregate Status
List File System Names

Restrictions

None.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
#define ZFSCALL_AGGR 0x40000005
#define AGOP_LISTAGGRNAMESZ_PARMMDATA 140 /* list attached aggregates with system name */
#define E2BIG 145

typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

#define ZFS_MAX_AGGNAME 44
#define SYS_MAX_NAMELEN 8 /* Max. z/OS system name length*/

typedef struct aggr_id2_t {
    char aid_eye[4]; /* Eye Catcher */
#define AID_EYE "AGID"
    char aid_len; /* Length of this structure */
    char aid_ver; /* Version */
#define AID_VER_2 2 /* version 2 */
    char aid_name[ZFS_MAX_AGGNAME+1]; /* aggr name, null terminated */
    char aid_sysname(SYS_MAX_NAMELEN+1); /* system name, NULL terminated */
    char aid_reserved[24]; /* Reserved for the future */
} AGGR_ID2;

struct parmstruct {
    syscall_parmlist myparms; /* Real malloc'd structure will have an array of AGGR_ID2s here */
    long size;
    char systemname[9];
};

int main(int argc, char **argv) {
    int bpxrv;
    int bpxrc;
    int bpxrs;

    struct parmstruct myparmstruct;
    AGGR_ID2 *aggPtr;
    int agg_size = sizeof(AGGR_ID2);
    int buffer_size = sizeof(AGGR_ID2);
    struct parmstruct *myp = &myparmstruct;
    int mypsize;
    char *tempsystem;
    int count_aggrs, total_aggrs;

    myparmstruct.myparms.opcode = AGOP_LISTAGGRNAMESZ_PARMMDATA;
    myparmstruct.myparms.parms[0] = 0;
    myparmstruct.myparms.parms[1] = 0;
```
myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

BPX1PCT("ZFS", /* Aggregate operation */
    ZFSCALL_AGGR,  /* Length of Argument */
    syscall_parmlist, /* Pointer to Argument */
    &bpxrv, /* Pointer to Return_value */
    &bpxrc, /* Pointer to Return_code */
    &bpxrs); /* Pointer to Reason_code */

if (bpxrv < 0)
    {
        if (bpxrc == E2BIG)
            {
                buflen = myp->size; /* Get buffer size needed */
                mysize = buflen + sizeof(syscall_parmlist) + sizeof(long) + 9;
                myp = (struct parmstruct *) malloc ((long) mysize);
                memset(myp, 0, mysize);
                /* This next field should only be set if parms[3] is non-zero */
                /* systemp = (char *)myp + buflen + sizeof(syscall_parmlist) + sizeof(long); */
                /* strcpy(systemp,"DCEIMGVN"); */ /* set system to get lsaggr info from */
                myp->myparms.opcode = AGOP_LISTAGGRNAMES2_PARMDATA;
                myp->myparms.parms[0] = buflen;
                myp->myparms.parms[1] = sizeof(syscall_parmlist);
                myp->myparms.parms[2] = sizeof(syscall_parmlist) + buflen;
                myp->myparms.parms[3] = 0;
                /* Only specify a non-zero offset for the next field (parms[3]) if */
                /* you are running z/OS 1.7 and above, and */
                /* you want lsaggr aggregates owned on a single system */
                myp->myparms.parms[3] = sizeof(syscall_parmlist) + buflen + sizeof(long);
                myp->myparms.parms[4] = 0;
                myp->myparms.parms[5] = 0;
                myp->myparms.parms[6] = 0;
                BPX1PCT("ZFS", /* Aggregate operation */
                    mysize, /* Length of Argument */
                    (char *) myp, /* Pointer to Argument */
                    &bpxrv, /* Pointer to Return_value */
                    &bpxrc, /* Pointer to Return_code */
                    &bpxrs);
                if (bpxrv == 0)
                    {
                        total_aggrs = buflen/aggSize;
                        count_aggrs = 1;
                        for(aggPtr = (AGGR_ID2 *) &(myp->size) ; count_aggrs <= total_aggrs ;
                            aggPtr++, count_aggrs++)
                        {
                            if (strlen(aggPtr->aid_name) != 0)
                                printf("%-64.64s %-8.8s
", aggPtr->aid_name,
                                    aggPtr->aid_sysname);
                            else /* lsaggr names failed with large enough buffer */
                                {
                                    printf("Error on ls aggr with large enough buffer\n");
                                    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
                                    free(myp);
                                    return bpxrc;
                                }
                        }
                    }
                else /* error was not E2BIG */
                    {
                        printf("Error on ls aggr trying to get required size\n");
                        printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
                        free(myp);
                        return bpxrc;
                    }
            }
        else /* asking for buffer size gave rv = 0; maybe there are no aggregates */
            {
                if (myparmstruct.size == 0)
                    {
                        printf("No attached aggregates\n");
                    }
                else /* No, there was some other problem with getting the size needed */
                    {
                        printf("Error getting size required\n");
                    }
            }
        }
List File Information

Purpose

This subcommand lists detailed file or directory information. This API is an `w_pioctl` (BPX1PIO) call specifying a path name rather than a `pfsctl` (BPX1PCT) call specifying a file system name.

Format

BPX1PIO parameter list

- `Pathname_length` int
- `Pathname` char[1025]
- `Command` int 0x0000A901
- `Argument_length` int `sizeof(FOBJ_INFO)`
- `Argument` ptr to `FOBJ_INFO`
- `Return_value` 0
- `Return_code` 0
- `Reason_code` 0

- `hyper`
- `high` int
- `low` int

- `FOBJ_TIME`
  - `fo_seconds` hyper
  - `fo_microseconds` int
  - `unused` int

- `FOBJ_ACLINFO`
  - `fo_index` int
  - `fo_length` int

- `FOBJ_AUDIT`
  - `fo_read` char
  - `fo_write` char
  - `fo_exec` char
  - `fo_res1` char
  - `FO_NONE` 0
  - `FO_SUCC` 1
  - `FO_FAIL` 2

- `FO_SYSRES_NUM` 9

- `FOBJ_SYSINFO`
  - `fo_vnode` hyper
  - `fo_vntok` hyper
  - `fo_openwaiters` int
  - `fo_internalopens` int
  - `fo_readopens` int
  - `fo_writeopens` int
  - `fo_denyreads` short
  - `fo_denywrites` short
  - `fo_advdenyreads` short
  - `fo_advdenywrites` short
  - `fo_sysflags` char
    - `FO_SEQREAD` 1
    - `FO_SEQWRITE` 2
    - `FO_FSPVALID` 4
    - `FO_ANODEVALID` 8
    - `FO_SYMLINKVALID` 16
    - `FO_METAUPDATES` 32
    - `FO_REVOKE` 64
    - `FO_THRASH` 128
  - `fo_sysflags2` char
    - `FO_OWNER` 1
    - `fo_unused` char[2]
  - `fo_unscheduled` int
  - `fo_segments` int
  - `fo_dirtysegments` int
  - `fo_metaissued` int
  - `fo_metapending` int
  - `fo_rights` int
  - `fo_xmits` short
List File Information

Return value 0 if request is successful, -1 if it is not successful

Return code
EBUSY Aggregate containing file system is quiesced
EINTR ZFS is shutting down
EINVAL Invalid parameter list
List File Information

EMUSERR Internal error using an osi service
ENQENT No such file or directory exists

Reason code
0xEFnnxxxx See z/OS Distributed File Service Messages and Codes

Usage notes
- The aggregate must be mounted or attached.
- If you set fo_inflags to 1, only local data is retrieved. If you set fo_inflags to 0, both global and local data are retrieved.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required
The issuer requires lookup authority (x) to the directory and READ authority (r) to the file.

Related services
List Aggregate Status (Version 2)

Restrictions
None.

Examples

```c
#pragma linkage(BPX1GCW, OS)
#pragma linkage(BPX1PIO, OS)
extern void BPX1GCW(int, char *, int *, int *, int *);
extern void BPX1PIO(int, char *, int, int, void *, int *, int *, int *);
#include <stdio.h>
#include <time.h>
#define ZFS_IOCTL_FILEINFO 0x0000A901 /* zFS ioctl command to return detailed fileinfo */
#define hiszero(a) ((a).low == 0 && (a).high == 0)
define u_int unsigned int
#define uint16_t unsigned short
typedef struct hyper  /* This is a 64 bit integer to zFS */
{
    unsigned long high;
    unsigned long low;
} hyper;
/*****************************************************************************/
/* The FOBJ_INFO structure is used to contain the output of the fileinfo */
/* ioctl to provide detailed information for a singular object in a */
/* zFS file system. */
/*****************************************************************************/
typedef struct FOBJ_ACLINFO_t
{
    int fo_index;  /* Index into the anode table of the location of the ACL */
    int fo_length; /* Length of the ACL */
} FOBJ_ACLINFO;

typedef struct FOBJ_AUDIT_t
{
    char fo_read;  /* read auditing information */
    char fo_write; /* write auditing information */
    char fo_exec;  /* exec auditing information */
    char fo_res1;
#define FO_NONE 0 /* no auditing */
#define FO_SUCC 1 /* success auditing */
#define FO_FAIL 2 /* fail auditing */
} FOBJ_AUDIT;

typedef struct FOBJ_TIME_t
{
    hyper fo_seconds; /* number of seconds since epoch */
    int fo_microseconds; /* number of microseconds since epoch */
    int fo_tres1; /* unused */
} FOBJ_TIME;

typedef struct FOBJ_SYSINFO_t /* HEX displacement into FOBJ_INFO */
```
List File Information

```c
typedef struct fobj_info_t /* HEX displacement into FOBJ_INFO */
{
  char fo_eye[4]; /* 000 - Eye catcher */
#define FO_EYE "FOIN"
#define FO_EYE "POIN"
  short fo_len; /* 004 - Length of this structure */
#define FO_VER INITIAL 1 /* Initial version */
#define FO_SYSINFO_ONLY 1 /* Only the in-memory system information is being requested */
  int fo_inode; /* 008 - Inode of the object */
  int fo_unique; /* 00C - Uniquer of the object */
  hyper fo_vnode; /* 138 - Address of vnode in zFS kernel memory */
  hyper fo_vntok; /* 140 - Address of USS vnode in z/OS Unix address space */
  int fo_openwaiters; /* 148 - Number of tasks waiting to open file because blocked */
  int fo_internalopens; /* 14C - Number of internal opens on the file */
  int fo_readopens; /* 150 - Number of opens for read on the file */
  int fo_writeopens; /* 154 - Number of write opens */
  short fo_denyread; /* 158 - Number of deny-read opens */
  short fo_denywrite; /* 15A - Number of deny-write opens */
  short fo_advdenyread; /* 15C - Number of adv. deny read opens */
  short fo_advdenywrite; /* 15E - Number of adv. deny write opens */
  char fo_sysflags; /* 160 - Misc. information */
#define FO_SYSINFO_ONLY 1 /* Only the in-memory system information is being requested */
#endif
#define FO_SYMPER 8 /* System has security information cached for anode */
#define FO_ANODEVALID 8 /* System has posix attribute and disk location information cached */
#define FO_SYSRES_NUM 9
#define FO_SYSRES 32 /* Client has sent metadata updates to the server, and cannot directly read without */
a server sync */
#define FO_REVOKE 64 /* Revoke in progress */
#define FO_THRASH 128 /* Object is considered sysplex-thrashing and thrash resolution is in effect for file */
  char fo_sysflags2; /* 161 - Misc. information 2 */
#define FO_OWNER 1 /* This system is the owner of the file system */
  char fo_unused[2]; /* 162 - reserved */
#define FO_UNALLOCATED 0xFFFFFFFF /* This value means block is not allocated in fo_direct or fo_indirect slot */
  int fo_unscheduled; /* 164 - Number of dirty 4K pages in the user file cache that have not yet been written to disk */
  int fo_pending; /* 168 - Number of pending 4K pages in transit to disk */
  int fo_segments; /* 16C - Number of 64K segment structures in the user file cache for the file */
  int fo_diryssegments; /* 170 - Number of 64K segment structures that have dirty pages in the user file cache */
  int fo_metaissued; /* 174 - Number of in-progress IOs to disk that will require a metadata update to reflect */
  new data in the file */
  int fo_metapending; /* 178 - Number of queued metadata updates for file, for I0s completed to new data for the file */
  int fo_rights; /* 17C - Token rights held for object */
  short fo_waits; /* 180 - Number by in-progress transmissions from client to server for this file */
#define FO_OWNER_perms; /* 184 - Number of in-progress forwarded operations due to thrashing object */
  int fo_metabuffers; /* 188 - Number of buffers for file in the metadata cache - client only */
  int fo_dirtybuffers; /* 18C - Number of dirty metadata buffers in the metadata cache for object - server only */
  char fo_owner[9]; /* 18C - the name of the owner */
  char fo_localsys[9]; /* 195 - the name of the local system */
  char fo_pad[2]; /* 196 - pad */
#define FO_SYSRES_NUM 9
  int fo_sysres[FO_SYSRES_NUM]; /* 1A0 - Reserved for future use */
} FOBJ_SYSINFO;
```

Chapter 13. zFS application programming interfaces 247
List File Information

FOBJ_ACLINFO fo_access; /* 0A8 - ACL information for access acl of object */
FOBJ_ACLINFO fo_model; /* 0B0 - ACL information for directory model acl */
FOBJ_ACLINFO fo_fmodel; /* 0B8 - ACL information for file model acl */
FOBJ_AUDIT fo_user; /* 0C0 - User auditing information */
FOBJ_AUDIT fo_auditor; /* 0C4 - Auditor auditing information */
char fo_permbits; /* 0C8 - Sticky and other bits */
define FO_ISVTX 4 /* sticky bit on */
define FO_ISUID 2 /* setuid */
define FO_ISGID 1 /* setgid */
#define FO_Is Text /* contents are pure text indicator */
define FO_DEFERTAG 1 /* Defer tag set until first write */
define FO_FILEFMT 6; /* 0=NA 1=BIN 2=NL 3=CR 4= LF */
/* 5=CRLF 6=LFCR 7=CRNL 8=REC */
define FO_CCSD 12 /* hex ccsid */
define FO_SECLABEL[8]; /* seclabel of the object */
define FO_ENTRYCOUNT; /* Number of names in the directory, if this is a directory */
define FO_FDATAVERSION; /* Posix linkcount of the object */
define FO_DATAMAJOR; /* Data version for directory updates */
define FO_MINOR; /* USU attribute flags of object */
define FO_CREATIONVERIF 8; /* Creation verifier */
define FO_MAJOORMINOR[8]; /* Major/Minor number if object is a char special file */
define FO_TYPE; /* 0x4 - Object type */
define FO_DIR 1 /* object is directory */
define FO_FILE 2 /* object is a regular file */
define FO_LINK 3 /* object is a symlink */
define FO_FIFO 4 /* object is a fifo */
define FO_CHARSTORESPEC; /* object is a char special file */
define FO_FLAGS; /* 0Fs - Additional flag bits of object */
define FO_VERS 1 /* Object is a directory stored in new-fast format */
define FO_BROKEN 2 /* The tree structure of this new-fast format dir is broken */
define FO_CONVERTFAIL 4 /* Automatic conversion of the directory failed */
define FO_OFFSET; /* Offset into the physical block that contains the anode for object */
define FO_NODESIZE; /* Physical block in aggregate that contains the anode */
define FO_STATUSLEVEL; /* Directory status byte */
define FO_RES[3]; /* reserved */
define FO_RES3[14]; /* for future use */
define FO_SYSINFO fo_info; /* System based transient information */
}

FOBJ_INFO; /* 1C4 total length */

int main(int argc, char **argv)
{
    int bpixv;
    int bpixrc;
    int bpixrs;

    char parm_pathname[1024];
    char pathname[1024];

    char pathp = NULL;
    FOBJ_INFO Fobj;
    FOBJ_INFO *fo=&fobj;
    void *arg = (void *)fo;
    int arglen = sizeof(fobj);

    char buffer1[80];
    char buffer2[80];
    hyper bogusSignedTime;
    char rp;
    char *timep;
    char time1_string[30];
    char time2_string[30];

    char seclabel[9];
    char temp;

    if (argc < 2)
    {
        printf("Please specify a file or directory path name as a parameter\n\n");
        exit[1];
    }

    strcpy(parm_pathname, argv[1], sizeof(pathname));

    if (parm_pathname[0] == '/') /* if absolute path name */
    {
        pathp = parm_pathname;
        /* put ptr to pathname in pathp */
    }
    else
    /* if relative path name */
    {
        pathname[0] = 0;
        bpixv = 0;
        bpixrc = 0;
        bpixrs = 0;
        BPX1GCW(sizeof(pathname), pathname, &bpixv, &bpixrc, &bpixrs);
        /* get current working directory path */
        if (bpixv == -1)
        {
            printf("BPX1GCW call failed rc %u rsn %8.8Xn", bpixr, bpixrs);
            return bpixrc;
        }
        if ((strlen(pathname) + strlen(parm_pathname)) + 1) > sizeof(pathname): /* if name longer than maximum pathname */

248 z/OS V2R1.0 Distributed File Service zFS Administration
List File Information

```c
{ printf("directory path name too long - input name len %d plus cwd len %d for buffer size %d\n",
    strlen(parm_pathname), strlen(pathname), sizeof(pathname));
    return 121; /* EINVAL */
}
strcat(pathname, "/"); /* take the current working directory and append slash */
strcat(pathname, parm_pathname); /* than append the input relative path name */
pathp = pathname; /* put ptr to result in pathp */
}

bpxrc = 0;
bpxrv = 0;
bpxrs = 0;
memset((char *)&fobj;, 0x00 sizeof(fobj));
memcpy(&fobj.fo_eye;, FO_EYE, 4);
fobj.fo_len = sizeof(fobj);
fobj.fo_ver = FO_VER_INITIAL;

BPX1PIO(strlen(pathp), pathp, ZFSIOCTL_FILEINFO, arglen, arg, &bpxrv, &bpxrc, &bpxrs);
if (bpxrv < 0)
{
    printf("Error getting fileinfo for pathname %s\n", pathp);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else /* Return from fileinfo was successful */
{
    printf(" Object path: %s\n", pathp);
    printf(" Inode is %lu\n", fo->fo_inode);
    printf(" Length is %llu\n", fo->fo_length);

    /* Some common object information */
    printf(" Object type is %s\n", fo->fo_type == FO_DIR ? "DIR" :
        fo->fo_type == FO_FILE ? "FILE" :
        fo->fo_type == FO_LINK ? "LINK" :
        fo->fo_type == FO_CHARSPEC ? "CHARSPEC" : "??");

    /* Some directory object information */
    if (fo->fo_type == FO_DIR)
    {
        printf(" Directory version %u\n", 
            fo->fo_flags & FO_VERS ? 5 : 4);
    }
    printf("\n");
    return 0;
}
```

List File System Names

Purpose

This subcommand returns the names of the file systems contained in a specified aggregate on this system; the aggregate must be attached.
List File System Names

Format

```
syscall_parmlist
  opcode                  138 AGOP_LISTFSNAMES_PARMDATA
  parms[0]                offset to AGGR_ID
  parms[1]                buffer length or 0
  parms[2]                offset to buffer or 0
  parms[3]                offset to size
  parms[4]                0
  parms[5]                0
  parms[6]                0
  AGGR_ID
    aid_eye               char[4] "AGID"
    aid_len               char sizeof(AGGR_ID)
    aid_ver               char 1
    aid_name              char[45] "OMVS.PRV.AGGR001.LDS0001"
    aid_reserved          char[33] 0
  FS_ID[n]
    fsid_eye              char[4] "FSID"
    fsid_len              char sizeof(FS_ID)
    fsid_ver              char 1
    fsid_res1             char 0
    fsid_res2             char 0
    fsid_id
      high                unsigned long
      low                 unsigned long
    fsid_aggrname         char[45]
    fsid_name             char[45]
    fsid_reserved         char[32]
    fsid_reserved2        char[2]
  size                  long
Return_value  0 if request is successful, -1 if it is not successful

Return code
  EINVAL ZFS is shutting down
  EINVAL Invalid parameter list
  EMVSERR Internal error using an osi service
  ENOTATT Aggregate is not attached
  E2BIG List is too big for buffer supplied
Reason_code
  0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
```

Usage notes
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required
None.

Related services
- List Attached Aggregate Names
- List File System Status

Restrictions
None.

Examples
```c
#include <stdio.h>

#define ZFSALL_AGGR 0x40000005
#define AGOP_LISTFSNAMES_PARMDATA 138
#define E2BIG 145

typedef struct syscall_parmlist_t {
  int opcode; /* Operation code to perform */
  int parms[7]; /* Specific to type of operation. */
  /* provides access to the parms */
} syscall_parmlist

#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
```
List File System Names

Chapter 13. zFS application programming interfaces
List File System Names

List File System Names (Version 2)

Purpose

The List File System Names (Version 2) subcommand call is an aggregate operation that returns the names of the zFS file systems contained in a specified aggregate on this system and their corresponding z/OS UNIX file system names (if they are mounted). The aggregate specified must be attached.
List File System Names (Version 2)

Format

syscall_parmlist
opcode 144 AGOP_LISTFSNAMES_PARMDATA2
parms[0] offset to AGGR_ID
parms[1] buffer length or 0
parms[2] offset to size
parms[3] 0
parms[4] 0
parms[5] 0
parms[6] 0
AGGR_ID
aid_eye char[4] "AGID"
aid_len char sizeof(AGGR_ID)
aid_ver char 1
aid_name char[45] "OMVS.PRV.AGGR001.LDS0001"
aid_reserved char[33] 0
FS_ID2[n]
  Array of FS_ID2s (n can be zero)
  fsid_eye char[4] "FSID"
  fsid_len char sizeof(FS_ID2)
  fsid_ver char 2
  fsid_res1 char 0
  fsid_res2 char 0
  fsid_id high unsigned long
  low unsigned long
  fsid_aggrname char[45]
  fsid_name char[45]
  fsid_mtnname char[45]
  fsid_reserved char[49]
size long

Return_value 0 if request is successful, -1 if it is not successful

Return code
EINTR ZFS is shutting down
EINVAL Invalid parameter list
EMVSERY Internal error using an osi service
ENDENT Aggregate is not attached
E2BIG List is too big for buffer supplied

Reason code
0xEFnnxxxx See z/OS Distributed File Service Messages and Codes

Usage notes
- The version 2 List File System Names returns an array of FS_ID2s.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required
None.

Related services
- List Attached Aggregate Names
- List File System Status

Restrictions
When FS_ID2 is used, if you specify the z/OS UNIX file system name (fsid_mtnname), you cannot specify the zFS file system name (fsid_name) nor the aggregate name (fsid_aggrname).

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_LISTFSNAMES_PARMDATA2 144
#define E2BIG 145
```
typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
        /* provides access to the parms */
        /* parms[4]-parms[6] are currently unused*/
    syscall_parmlist;
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44

typedef struct aggr_id_t {
    char aid_eye[4]; /* Eye Catcher */
    #define AID_EYE "AGID"
    char aid_len; /* Length of this structure */
    #define AID_VER_INITIAL 1 /* Initial version */
    char aid_name[ZFS_MAX_AGGRNAME+1]; /* aggr name, null terminated */
    char aid_reserved[33]; /* Reserved for the future */
} AGGR_ID;

typedef struct hyper { /* This is a 64 bit integer to zFS */
    unsigned long high;
    unsigned long low;
} hyper;

typedef struct fs_id2_t {
    char fsid_eye[4]; /* Eye catcher */
    #define FSID_EYE "FSID"
    char fsid_len; /* Length of this structure */
    char fsid_ver; /* Version */
    char fsid_res1; /* Reserved. */
    char fsid_res2; /* Reserved. */
    hyper fsid_id; /* Internal identifier */
    #define FSID_VER_2 2 /* Second version */
    char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string */
    char fsid_name[ZFS_MAX_FSYSNAME+1]; /* Name, null terminated */
    char fsid_mtnname[ZFS_MAX_FSYSNAME+1]; /* Mount name, null terminated */
    char fsid_reserved[49]; /* Reserved for the future */
} FS_ID2;

typedef struct parmstruct
{
    syscall_parmlist myparms;
    AGGR_ID aggr_id;
    /* Real malloc'd structure will have an array of FS_ID2s here */
} long size;

int main(int argc, char **argv)
{
    int bxrv;
    int bxrc;
    int bxrs;

    struct parmstruct myparmstruct;
    AGGR_ID *aggPtr;
    /* Real malloc'd structure will have an array of FS_ID2s here */
    int fsize = sizeof(FS_ID2);
    int buflen = sizeof(FS_ID2);
    struct parmstruct *myp = &myparmstruct;
    int mypsize;
    int count_fs, total_fs;

    char aggrname[45]="OMVS.PRV.AGGR001.LDS0001";

    memset(&myparmstruct.aggr_id,0,sizeof(AGGR_ID)); /* Ensure reserved fields are 0 */

    memcpy(&myparmstruct.aggr_id.aid_eye,AID_EYE,4);
    myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
    myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
    strcpy(myparmstruct.aggr_id.aid_name, aggrname);

    myparmstruct.myparms.opcode = AGOP_LISTFSNAMES_PARMDATA2;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = 0;
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    BPX1PCT("ZFS ",
        ZFSCALL_AGGR, /* Aggregate operation */
        sizeof(myparmstruct), /* Length of Argument */
        (char) *myparmstruct, /* Pointer to Argument */
        &bxrv, /* Pointer to Return_value */
        &bxrc, /* Pointer to Return_code */
        &bxrs); /* Pointer to Reason_code */

    if (bxrv < 0)
if (bpxrc == E2BIG)
{
    buflen = myp->size; /* Get buffer size needed */
    mypsize = buflen + sizeof(syscall_parmlist) + sizeof(AGGR_ID) +
              sizeof(myparmstruct.size);
    myp = (struct parmstruct *) malloc ((long) mypsize);
    memset(myp, 0, mypsize);
    memcpy(myp->aggr_id.aid_eye, AID_EYE, 4);
    myp->aggr_id.aid_len = sizeof(AGGR_ID);
    myp->aggr_id.aid_ver = AID_VER_INITIAL;
    strcpy(myp->aggr_id.aid_name, aggrname);
    myp->myparms.opcode = AGOP_LISTFSNAMES_PARMDATA2;
    myp->myparms.parms[0] = sizeof(syscall_parmlist);
    myp->myparms.parms[1] = buflen;
    myp->myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
    myp->myparms.parms[3] = sizeof(syscall_parmlist) + sizeof(AGGR_ID) + buflen;
    myp->myparms.parms[4] = 0;
    myp->myparms.parms[5] = 0;
    myp->myparms.parms[6] = 0;

    BPX1PCT("ZFS ", /* Aggregate operation */
        ZFSCALL_AGGR, /* Aggregate operation */
        mypsize, /* Length of Argument */
        (char *) myp, /* Pointer to Argument */
        &bpxrv, /* Pointer to Return_value */
        &bpxrc, /* Pointer to Return_code */
        &bpxrs); /* Pointer to Reason_code */
    if (bpxrv == 0)
    {
        total_fs = buflen/fsSize;
        printf("total file systems = %d in aggregate %s\n",total_fs, aggrname);
        count_fs = 1;
        for(fsPtr = (FS_ID2 *) &(myp->size) ; count_fs <= total_fs ; fsPtr++, count_fs++)
        {
            printf("\n");
            printf("zFS file system name [%s]\n",fsPtr->fsid_name);
            printf("UNIX file system name [%s]\n",fsPtr->fsid_mtnname);
        }
        free(myp);
    }
    else /* lsaagr names failed with large enough buffer */
    {
        printf("Error on lsaagr names failed with large enough buffer\n");
        printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        free(myp);
        return bpxrc;
    }
    else /* error was not E2BIG */
    {
        printf("Error on lsaagr names failed with large enough buffer\n");
        printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        free(myp);
        return bpxrc;
    }
    else /* asking for buffer size gave rv = 0; maybe there are no file system */
    {
        if (myparmstruct.size == 0)
        {
            printf("No file systems\n");
        }
        else /* No, there was some other problem with getting the size needed */
        {
            printf("Error getting size required\n");
        }
    }
    return 0;
}

List File System Status

Purpose

This subcommand lists status information of a file system. As input, use an FS_ID
or an FS_ID2, which specifies the z/OS UNIX file system name (the mount name).
For an FS_ID2, the file system must be mounted using that z/OS UNIX file system
name. The aggregate containing the file system must be attached; also, the
aggregate cannot be quiesced.
List File System Status

Format

```c
syscall_parmlist
 opcode 142 FSOP_GETSTAT_PARMDATA
 parms[0] offset to FS_ID
 parms[1] offset to FS_STATUS
 parms[2] 0
 parms[3] 0
 parms[4] 0
 parms[5] 0
 parms[6] 0
 FS_ID or FS_ID2

  fsid_eye char[4] "FSID"
  fsid_len char sizeof(FS_ID)
  fsid_ver char 1
  fsid_res1 char 0
  fsid_res2 char 0
  fsid_aggrname char[45] 0
  fsid_name char[45] "OMVS.PRV.FS3"
  fsid_reserved char[32] 0
  fsid_reserved2 char[2] 0

 FS_ID2 or FS_ID

  fsid_eye char[4] "FSID"
  fsid_len char sizeof(FS_ID2)
  fsid_res1 char 0
  fsid_res2 char 0
  fsid_aggrname char[45] 0
  fsid_name char[45] "OMVS.PRV.FS3"
  fsid_res1 char 0
  fsid_res2 char 0
  fsid_aggrname char[45] 0
  fsid_name char[45] "OMVS.PRV.MNT.FS3"
  fsid_reserved char[49] 0
  fsid_reserved2 char[2] 0
  fs_status
   fs_eye char[4] "FSST"
   fs_len short sizeof(FS_STATUS)
   fs_ver char 1
   fs_res1 char 0
   fs_aggrname char[45] 0
   fs_aggrname char[45] 0
   fs_id char[4] "FSST"
   fs_id char[4] "FSST"
   fs_id char[4] "FSST"
   fs_id char[4] "FSST"
   fs_id char[4] "FSST"

Continued...
```
List File System Status

Continued...

fsid_aggrname char[45] 0
fsid_name char[45] 0
fsid_mntname char[45] "OMVS.PRV.MNT.FS3"
fsid_reserved char[49] 0
FS_STATUS
  fs_eye char[4] "FSST"
  fs_len short sizeof(FS_STATUS)
  fs_ver char 1
  fs_res1 char 0
  fs_id
    high unsigned long 0
    low unsigned long 0
  fs_cloneTime timeval 0
  fs_createTime timeval 0
  fs_updateTime timeval 0
  fs_accessTime timeval 0
  fs_allocLimit unsigned long 0
  fs_allocUsage unsigned long 0
  fs_visQuotaLimit unsigned long 0
  fs_visQuotaUsage unsigned long 0
  fs_accError unsigned long 0
  fs_accStatus unsigned long 0
  fs_states long 0
  fs_inodeMax long 0
  fs_minQuota long 0
  fs_type long 0
  fs_threshold char 0
  fs_increment char 0
  fs_mountstate char 0
  FS_NOT_MOUNTED 0
  FS_MOUNTED_RW 1
  FS_MOUNTED_RO 2
  fs_msglen char 0
  fs_msg char[128] 0
  fs_aggrname char[45] 0
  fs_reserved1 char[3]
  fs_reserved2 unsigned long[3] 0
  fs_InodeTbl unsigned long 0
  fs_requests
    high unsigned long 0
    low unsigned long 0
  fs_res2 unsigned long 0
  fs_res3 unsigned long 0
  fs_res4 unsigned long 0
  fs_res5 unsigned long 0
  fs_pad1 int
  fs_diskFormatMajorVersion unsigned long 0
  fs_diskFormatMinorVersion unsigned long 0
  fs_reserved char[80]

- OR -

-FS_STATUS2
  fs_eye char[4] "FSST"
  fs_len short sizeof(FS_STATUS)
  fs_ver char 2
  fs_res1 char 0
  fs_id
    high unsigned long 0
    low unsigned long 0
  fs_cloneTime timeval 0
  fs_createTime timeval 0
  fs_updateTime timeval 0
  fs_accessTime timeval 0
  fs_allocLimit unsigned long 0
  fs_allocUsage unsigned long 0
  fs_visQuotaLimit unsigned long 0
  fs_visQuotaUsage unsigned long 0
  fs_accError unsigned long 0
  fs_accStatus unsigned long 0
  fs_states long 0
  fs_inodeMax long 0
  fs_minQuota long 0
  fs_type long 0
  fs_threshold char 0
  fs_increment char 0
  fs_mountstate char 0
  FS_NOT_MOUNTED 0
  FS_MOUNTED_RW 1
  FS_MOUNTED_RO 2
  fs_msglen char 0
  fs_msg char[128] 0
  fs_aggrname char[45] 0
  fs_reserved1 char[3]
  fs_reserved2 unsigned long[3] 0
  fs_InodeTbl unsigned long 0
  fs_requests
    high unsigned long 0
    low unsigned long 0
Usage notes

- The aggregate must be mounted (or attached).
- For an FS_STATUS, if a size is too large for 32 bits, 0xFFFFFFFF will be returned. For an FS_STATUS2, sizes are returned in both the normal fields and the hyper fields.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

List Attached Aggregate Names

Restrictions

When FS_ID2 is used, if you specify the z/OS UNIX file system name (fsid_mntname), you cannot specify the zFS file system name (fsid_name) nor the aggregate name (fsid_aggrname).

The following fields are internal use only and not intended for application use:

- fs_accError
- fs_accStatus
- fs_type

The fs_states field contains flag 0x00010000, indicating a read/write file system, and flag 0x00030000, indicating a backup file system. All other flags in this field are internal use only and are not intended for application usage.

Examples

Example 1: The following example uses an FS_ID; see Example 2 for an example that uses FS_ID2.

```c
#include <stdio.h>
#include <ctime>

#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#define ZFSCALL_FILESYS 0x40000004
#define FSOP_GETSTAT_PARMDATA 142
```
typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused */
} syscall_parmlist;

typedef struct hyper { /* This is a 64 bit integer to zFS */
    unsigned long high;
    unsigned long low;
} hyper;

#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44

typedef struct fs_id_t { /* Eye catcher */
    char fsid_eye[4]; /* Eye catcher */
    #define FSID_EYE "FSID"
    char fsid_len; /* Length of this structure */
    char fsid_ver; /* Version */
    char fsid_res1; /* Reserved. */
    char fsid_res2; /* Reserved. */
    hyper fsid_id; /* Internal identifier */
    #define FSID_VER_INITIAL 1 /* Initial version */
    char fsid_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string */
    char fsid_name[ZFS_MAX_FSYSNAME+1]; /* Name, null terminated */
    char fsid_reserved[32]; /* Reserved for the future */
    char fsid_reserved[2]; /* Reserved for the future */
} FS_ID;

typedef unsigned long u_long;

struct timeval {
    long tv_sec; /* seconds */
    long tv_usec; /* microseconds */
};

typedef struct fs_status_t { /* Eye catcher */
    char fs_eye[4]; /* Eye catcher */
    #define FS_EYE "FSST"
    short fs_len; /* Length of structure */
    char fs_ver; /* Version */
    #define FS_VER_INITIAL 1 /* Initial version */
    char fs_flags; /* Flags */
    #define FS_PERFINFO 0x80 /* Performance information in output status */
    hyper fs_id; /* Internal identifier */
    struct timeval fs_cloneTime; /* Time when this filesystem made via clone or when last recloned */
    struct timeval fs_createTime; /* Time when this filesystem was created */
    struct timeval fs_updateTime; /* Time when this filesystem was last updated */
    struct timeval fs_accessTime; /* Time when this filesystem was last accessed */
    u_long fs_allocLimit; /* Allocation limit for filesystems in kilobytes */
    u_long fs_allocUsage; /* Allocation used for filesystems in kilobytes */
    u_long fs_visQuotaLimit; /* Visible filesystem quota in kilobytes */
    u_long fs_visQuotaUsage; /* How much quota is used in kilobytes */
    u_long fs_accError; /* Error to return for incompatible vnode ops */
    long fs_accStatus; /* Operations currently being performed on file system */
    long fs_states; /* State bits */
    #define FS_TYPE_RW 0x10000 /* read/write (ordinary) */
    #define FS_TYPE_BK 0x30000 /* ``.backup'' */
    long fs_nodeMax; /* Maximum node number used */
    long fs_minQuota; /* Threshold for fsfull monitoring */
    char fs_increment; /* Increment for fsfull monitoring */
    char fs_mountstate; /* Filesystem not mounted */
    #define FS_MOUNTEO 0 /* Filesystem not mounted */
    #define FS_MOUNTED_RW 1 /* Filesystem mounted RW */
    #define FS_MOUNTED_RW 2 /* Filesystem mounted RD */
    char fs_mglen; /* Length of status message */
    char fs_msg[256]; /* Status message for filesystem */
    char fs_aggrname[ZFS_MAX_AGGRNAME+1]; /* Name of aggregate I reside on */
    char fs_reserved[3]; /* Reserved for future use/alignment */
    u_long fs_reserved[3]; /* Reserved */
    u_long fs_mode; /* Amount of k used for the filesystem Inode table */
    /* fs_mode is zero for all releases prior to r7 and non zero in r7 and above */
    hyper fs_requests; /* Number of filesystem requests by users/applications */
    u_long fs_reserved3;
    u_long fs_reserved4;
    u_long fs_reserved5;
    u_long fs_reserved6;
    u_long fs_reserved7;
    u_long fs_diskFormatMajorVersion; /* disk format major version */
    u_long fs_diskFormatMinorVersion; /* disk format minor version */
    char fs_reserved[80]; /* Reserved for future use */
} FS_STATUS;

struct parmstruct { /* sysexec_parmlist myparms */
    syscall_parmlist myparms;

List File System Status

Chapter 13. zFS application programming interfaces 259
List File System Status

```c
int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char filesystemname[45] = "OMVS.PRV.FS3"; /* file system name to getstatus */
    struct parmstruct myparmstruct;
    FS_ID *idp = &(myparmstruct.fs_id);
    FS_STATUS *fsp = &(myparmstruct.fs_status);

    myparmstruct.myparms.opcode = FSOP_GETSTAT_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(FS_ID);
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;
    memset(idp,0,sizeof(FS_ID)); /* Ensure reserved fields are 0 */
    memset(fsp,0,sizeof(FS_STATUS)); /* Ensure reserved fields are 0 */
    memcpy(&myparmstruct.fs_status.fs_eye[0], FS_EYE, 4);
    myparmstruct.fs_status.fs_len = sizeof(FS_STATUS);
    myparmstruct.fs_status.fs_ver = FS_VER_INITIAL;
    memcpy(&myparmstruct.fs_id.fsid_eye,FSID_EYE,4);
    myparmstruct.fs_id.fsid_len = sizeof(FS_ID);
    myparmstruct.fs_id.fsid_ver = FSID_VER_INITIAL;
    strcpy(myparmstruct.fs_id.fsid_name,filesystemname);

    BPX1PCT("ZFS ", /* File system operation */
        sizeof(myparmstruct), /* Length of Argument */
        (char *) &myparmstruct, /* Pointer to Argument */
        &bpxrv, /* Pointer to Return_value */
        &bpxrc, /* Pointer to Return_code */
        &bpxrs); /* Pointer to Reason_code */

    if (bpxrv < 0)
    {
        printf("Error getstatus file system %s\n", filesystemname);
        printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    }
    else /* Return from getstatus was successful */
    {
        printf("File system %s getstatus successful\n",filesystemname);
        printf("getstatus: fs_id=%d,,%d, clone_time=%s, create_time=%s, update_time=%s, access_time=%s\n",
            myparmstruct.fs_status.fs_id.high,
            myparmstruct.fs_status.fs_id.low,
            ctime(&myparmstruct.fs_status.fs_cloneTime.tv_sec),
            ctime(&myparmstruct.fs_status.fs_createTime.tv_sec),
            ctime(&myparmstruct.fs_status.fs_updateTime.tv_sec),
            ctime(&myparmstruct.fs_status.fs_accessTime.tv_sec));
        printf("getstatus: alloc_limit=%u, alloc_usage=%u, quota_limit=%u\n",
            myparmstruct.fs_status.fs_allocLimit,
            myparmstruct.fs_status.fs_allocUsage,
            myparmstruct.fs_status.fs_visQuotaLimit);
        printf("getstatus: quota_usage=%u, accError=%u, accStatus=%x, states=%x\n",
            myparmstruct.fs_status.fs_visQuotaUsage,
            myparmstruct.fs_status.fs_accError,
            myparmstruct.fs_status.fs_accStatus,
            myparmstruct.fs_status.fs_states);
        printf("getstatus: max_inode=%d, min_quota=%d, type=%d, fsfull_threshold=%d\n",
            myparmstruct.fs_status.fs_nodeMax,
            myparmstruct.fs_status.fs_minQuota,
            myparmstruct.fs_status.fs_type,
            myparmstruct.fs_status.fs_threshold);
        printf("getstatus: fsfull_increment=%ld, mount_state=%d, msg_len=%d, msg=%s\n",
            myparmstruct.fs_status.fs_increment,
            myparmstruct.fs_status.fs_mountstate,
            myparmstruct.fs_status.fs_msglen,
            myparmstruct.fs_status.fs_msg);
        printf("getstatus: aggrname=%s\n", myparmstruct.fs_status.fs_aggrname);
        printf("getstatus: inode_table_k=%d, fs_requests=%d,,%d\n",
            myparmstruct.fs_status.fs_inodeTableK,
            myparmstruct.fs_status.fs_requests.high,
            myparmstruct.fs_status.fs_requests.low);
        printf("getstatus: version=%d,fsidname=%s\n",
            myparmstruct.fs_status.fs_diskFormatMajorVersion,
            myparmstruct.fs_status.fs_diskFormatMinorVersion);
        return 0;
    }
}
```
Example 2: The following example uses FS_ID2; see Example 1 for an example that uses FS_ID.

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *
#include <stdio.h>
#include <time.h> /* ctime */
#define ZFSCALL_FILESYS 0x40000004
#define FSOP_GETSTAT_PARMDATA 142
typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;
typedef struct hyper { /* This is a 64 bit integer to zFS */
    unsigned long high;
    unsigned long low;
} hyper;
#define ZFS_MAX_AGGRNAME 44
#define ZFS_MAX_FSYSNAME 44
typedef struct fs_id2_t {
    char fsid_eye[4]; /* Eye catcher */
    #define FSID_EYE "FSID"
    char fsid_len; /* Length of this structure */
    char fsid_ver; /* Version */
    char fsid_res1; /* Reserved. */
    char fsid_res2; /* Reserved. */
    hyper fsid_id; /* Internal identifier */
    #define FSID_VER_2 2 /* version for FS_ID2 */
    char fsid_apgrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string */
    char fsid_name[ZFS_MAX_FSYSNAME+1]; /* Name, null terminated */
    char fsid_reserved[49]; /* Reserved for the future */
} FS_ID2;
typedef unsigned long u_long;
struct timeval {
    long tv_sec; /* seconds */
    long tv_usec; /* microseconds */
};
typedef struct fs_status_t {
    char fs_eye[4]; /* Eye catcher */
    #define FS_EYE "FSST"
    short fs_len; /* Length of structure */
    char fs_ver; /* Initial version */
    char fs_flags; /* Flags */
    #define FS_PERFINFO 0x80 /* Performance information in output status */
    hyper fs_id; /* Internal identifier */
    struct timeval fs_cloneTime; /* Time when this filesys made via clone or when last recloned */
    struct timeval fs_createTime; /* Time when this filesys was created */
    struct timeval fs_updateTime; /* Time when this filesys was last updated */
    struct timeval fs_accessTime; /* Time when this filesys was last accessed */
    u_long fs_allocLimit; /* Allocation limit for filesys in kilobytes*/
    u_long fs_visQuotaLimit; /* Visible filesystem quota in kilobytes*/
    u_long fs_visQuotaUsage; /* How much quota is used in kilobytes*/
    u_long fs_accError; /* error to return for incompatible vnode ops*/
    long fs_accStatus; /* Operations currently being performed on file system */
    long fs_states; /* State bits */
    #define FS_TYPE_RW 0x10000 /* read/write (ordinary) */
    #define FS_TYPE_BK 0x30000 /* .backup */
    long fs_type; /* Maximum inode number used */
    long fs_type; /* Threshold for fsfull monitoring */
    char fs_minQuota; /* Increment for fsfull monitoring */
    char fs_minQuota; /* Aggregate flags */
    #define FS_NOT_MOUNTED 0 /* Files not mounted */
    #define FS_MOUNTED_RW 1 /* Files mounted RW */
    #define FS_MOUNTED_RDONLY 2 /* Filesys mounted RO */
    char fs_mglen; /* Length of status message */
    char fs_mglen[128]; /* Status message for filesystem */
    char fs_apgrname[ZFS_MAX_AGGRNAME+1]; /* Name of aggregate I reside on */
    char fs_reserved[3]; /* Reserved for future use/alignment */
    unsigned long fs_reserved[2]; /* Amount of k used for the Filesystem Inode table*/
    hyper fs_requests; /* Number of filesystem requests by users/applications */
    u_long fs_reserved3;
    u_long fs_reserved4;
```
u_long    fs_reserved5;
int       fs_pad1;
uint long fs_diskFormatMajorVersion; /* disk format major version */
uint long fs_diskFormatMinorVersion; /* disk format minor version */
char      fs_reserved[80]; /* Reserved for future use */
}

struct parmstruct
{
    syscall_parmlist myparms;
    FS_ID2  fs_id2;
    FS_STATUS fs_status;
};

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char filesystemname[45] = "OMVS.PRV.MNT.FS3"; /* file system name to getstatus */

    struct parmstruct myparmstruct;
    FS_ID2 *idp = &(myparmstruct.fs_id2);
    FS_STATUS *fsp = &(myparmstruct.fs_status);

    myparmstruct.myparms.opcode = FSOP_GETSTAT_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(FS_ID2);
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    memset(idp,0,sizeof(FS_ID2));  /* Ensure reserved fields are 0 */
    memset(fsp,0,sizeof(FS_STATUS));  /* Ensure reserved fields are 0 */

    memcpy(&myparmstruct.fs_status.fs_eye[0], FS_EYE, 4);
    myparmstruct.fs_status.fs_len = sizeof(FS_STATUS);
    myparmstruct.fs_status.fs_ver = FS_VER_INITIAL;

    memcpy(&myparmstruct.fs_id2.fsid_eye,FSID_EYE,4);
    myparmstruct.fs_id2.fsid_len = sizeof(FS_ID2);
    strcpy(myparmstruct.fs_id2.fsid_mtname,filesystemname);

    BPX1PCT("ZFS ",
            ZFSCALL_FILESYS, /* File system operation */
            sizeof(myparmstruct), /* Length of Argument */
            (char *) &myparmstruct, /* Pointer to Argument */
            &bpxrv, /* Pointer to Return_value */
            &bpxrc, /* Pointer to Return_code */
            &bpxrs); /* Pointer to Reason_code */

    if (bpxrv < 0)
    {
        printf("Error getstatus file system %s\n", filesystemname);
        printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    }
    else /* Return from getstatus was successful */
    {
        printf("File system %s getstatus successful\n",filesystemname);
        printf("getstatus: fs_id=%d,,%d, clone_time=%s, create_time=%s, update_time=%s, access_time=%s\n",
               myparmstruct.fs_status.fs_id.high,
               myparmstruct.fs_status.fs_id.low,
               ctime(&myparmstruct.fs_status.fs_cloneTime.tv_sec),
               ctime(&myparmstruct.fs_status.fs_createTime.tv_sec),
               ctime(&myparmstruct.fs_status.fs_updateTime.tv_sec),
               ctime(&myparmstruct.fs_status.fs_accessTime.tv_sec));
        printf("getstatus: alloc_limit=%u, alloc_usage=%u, quota_limit=%u\n",
               myparmstruct.fs_status.fs_allocLimit,
               myparmstruct.fs_status.fs_allocUsage,
               myparmstruct.fs_status.fs_visQuotaLimit);
        printf("getstatus: quota_usage=%u, accError=%u, accStatus=%x, states=%x\n",
               my parmstruct.fs_status.fs_visQuotaUsage,
               myparmstruct.fs_status.fs_accError,
               myparmstruct.fs_status.fs_accStatus,
               myparmstruct.fs_status.fs_states);
        printf("getstatus: max_inode=%d, min_quota=%d, type=%d, fsfull_threshold=%d\n",
               myparmstruct.fs_status.fs_nodeMax,
               myparmstruct.fs_status.fs_minQuota,
               myparmstruct.fs_status.fs_type,
               myparmstruct.fs_status.fs_threshold);
        printf("getstatus: fsfull_increment=%d, mount_state=%d, msg_len=%d, msg=%s\n",
               myparmstruct.fs_status.fs_increment,
               myparmstruct.fs_status.fs_mountstate,
               myparmstruct.fs_status.fs_mslen,
               myparmstruct.fs_status.fs_msg);
    }
List File System Status

printf("getstatus: aggrname=%s\n", myparmstruct.fs_status.fs_aggrname);
printf("getstatus: inode_table_k=%d, fs_requests=%d,,%d
", 
  myparmstruct.fs_status.fs_InodeTbl,
  myparmstruct.fs_status.fs_requests.high,
  myparmstruct.fs_status.fs_requests.low);
printf("getstatus: version=%d.%d
", 
  myparmstruct.fs_status.fs_diskFormatMajorVersion,
  myparmstruct.fs_status.fs_diskFormatMinorVersion);
}
return 0;
}

List Systems

Purpose

This subcommand call is used to retrieve the system names that are part of the zFS XCF group.

Format

<table>
<thead>
<tr>
<th>syscall_parmlist</th>
<th>opcode 174 CFGOP_LSSYS</th>
</tr>
</thead>
<tbody>
<tr>
<td>parms[0]</td>
<td>size of buffer</td>
</tr>
<tr>
<td>parms[1]</td>
<td>offset to buffer</td>
</tr>
<tr>
<td>parms[2]</td>
<td>offset to size</td>
</tr>
<tr>
<td>parms[3]</td>
<td>0</td>
</tr>
<tr>
<td>parms[4]</td>
<td>0</td>
</tr>
<tr>
<td>parms[5]</td>
<td>0</td>
</tr>
<tr>
<td>parms[6]</td>
<td>0</td>
</tr>
<tr>
<td>buffer</td>
<td>char[ ]</td>
</tr>
<tr>
<td>size</td>
<td>int</td>
</tr>
</tbody>
</table>

Return_value 0 if request successful, -1 if it is not successful

Return_code

- E2BIG Data to return is too large for buffer supplied
- EINTR ZFS is shutting down
- EMVSERR Internal error
- ERANGE No systems to return

Reason_code

- OxEFnnxxx See z/OS Distributed File Service Messages and Codes

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

- Query sysplex_state

Restrictions

None.

Examples

#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *, int *);
#include <stdio.h>
#define ZFSCALL_CONFIG 0x40000006 174 /* List names of systems in the sysplex */
#define CFGOP_LSSYS
#define E2BIG 145 /* data to return is too big for buffer */
#define ERANGE 2 /* there were no systems to return */
List Systems

typedef struct system_name_t {
    char sys_name[9]; /* 8 byte name, null terminated */
} SYSTEM_NAME;

typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

struct parmstruct {
    syscall_parmlist myparms;
    /* SYSTEM_NAME buffer[32]; */ /* output buffer for sysnames */
    int size;
} myparmstruct;

int main(int argc, char **argv) {
    int bpxrv;
    int bpxrc;
    int bpxrs;
    int i;

    struct parmstruct *myp = &myparmstruct;
    int mypsize, buflen;

    myparmstruct.myparms.opcode = CFGOP_LSSYS;
    myparmstruct.myparms.parms[0] = 0; /* size of buffer */
    myparmstruct.myparms.parms[1] = 0; /* offset to buffer */
    myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist); /* offset to size (required size) */
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    BPIXPT("ZFS ",
        ZFSCALL_CONFIG, /* Config query operation */
        sizeof(myparmstruct), /* Length of Argument */
        (char *) &myparmstruct, /* Pointer to Argument */
        &bpxrv, /* Pointer to Return_value */
        &bpxrc, /* Pointer to Return_code */
        &bpxrs); /* Pointer to Reason_code */

    if( bpxrv < 0 ) {
        if( bpxrc == E2BIG ) {
            buflen = myparmstruct.size; /* Get buffer size needed */
            mypsize = sizeof(myparmstruct) + buflen + sizeof(myparmstruct.size);
            myp = (struct parmstruct *) malloc ((long) mypsize);
            memset(myp, 0, mypsize);

            myp->myparms.opcode = CFGOP_LSSYS;
            myp->myparms.parms[0] = buflen; /* size of buffer */
            myp->myparms.parms[1] = 0; /* offset to buffer */
            myp->myparms.parms[2] = sizeof(syscall_parmlist); /* offset to size (required size) */
            myp->myparms.parms[3] = 0;
            myp->myparms.parms[4] = 0;
            myp->myparms.parms[5] = 0;
            myp->myparms.parms[6] = 0;

            BPIXPT("ZFS ",
                ZFSCALL_CONFIG, /* Config query operation */
                mypsize, /* Length of Argument */
                (char *) myp, /* Pointer to Argument */
                &bpxrv, /* Pointer to Return_value */
                &bpxrc, /* Pointer to Return_code */
                &bpxrs); /* Pointer to Reason_code */
        }
    }

    if( bpxrv == 0 ) {
        int j,syscount;
        SYSTEM_NAME *syslist;
        int *sizep;

        sizep=(int *)((int)myp + sizeof(syscall_parmlist) + buflen);
        syslist=(SYSTEM_NAME *)((int)myp + sizeof(syscall_parmlist));
        syscount=(*sizep)/sizeof(SYSTEM_NAME);
        for (j=1; j <= syscount; j++) {
            printf("%-8.8s
", syslist->sys_name);
            syslist++;
        }
    }
    free(myp);
}
else /* lssys failed with large enough buffer */
{
  if( bpxrc == ERANGE )
  {
    printf("No systems to display\n");
  }
  else
  {
    printf("Error on lssys with large enough buffer\n");
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    free(myp);
    return bpxrc;
  }
}
else /* error was not E2BIG on the original BPX1PCT */
{
  if( bpxrc == ERANGE )
  {
    printf("No systems to display from original BPX1PCT\n");
  }
  else
  {
    printf("Error on lssys trying to get required size\n");
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    return bpxrc;
  }
}
else /* asking for buffer size gave rv = 0; maybe there is no data */
{
  if( myparmstruct.size == 0 )
  {
    printf("No data\n");
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
  }
  else /* No, there was some other problem with getting the size needed */
  {
    printf("Error getting size required\n");
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
  }
  return 0;
}

Query Config Option

Purpose

Query Config Option is a set of subcommand calls (configuration operations) that retrieve the current value for a particular configuration setting. Each one returns the configuration setting as a character string in the co_string field.

The following Format and Example for query admin threads use the CFGOP_QUERY_ADM_THREADS subcommand. Example for syslevel is an example to query the syslevel. The other query subcommands (see Table 10 on page 209) operate in a similar manner.
Query Config Option

Format

```c
syscall_parmlist
  opcode 180 CFGOP_QUERY_ADM_THREADS
  parms[0] offset to CFG_OPTION
  parms[1] offset to system name (optional)
  parms[2] 0
  parms[3] 0
  parms[4] 0
  parms[5] 0
  parms[6] 0

CFG_OPTION
  co_eye char[4] "CFOP"
  co_len short sizeof(CFG_OPTION)
  co_ver char 1
  co_string char[81] 0
  co_value int[4] 0
  co_reserved char[24] 0
  systemname char[9]

Return_value 0 if request is successful, -1 if it is not successful

Return_code
  EBUSY Aggregate could not be quiesced
  EINTR ZFS is shutting down
  EMVSERR Internal error using an osi service
  ENOENT Aggregate is not attached
  EPERM Permission denied to perform request

Reason_code
  0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

Set Config Option

Restrictions

None.

Examples

Example 1: The following example shows an API to query admin threads.

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>
define ZFSCALL_CONFIG 0x40000006
define CFGOP_QUERY_ADM_THREADS 180 /* query number of admin threads */
typedef struct syscall_parmlist_t {
  int opcode; /* Operation code to perform */
  int parms[7]; /* Specific to type of operation. */
  /* provides access to the parms */
  /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct config_option_t {
  char co_eye[4] /* Eye catcher */
  char co_reserved[24] 0
  char co_string[81] 0
  int co_value[4] 0
  int co_len short sizeof(CFG_OPTION)
  char co_ver char 1
  char co_eye[4] "CFOP"
} config_option_t;
```

Return_value 0 if request is successful, -1 if it is not successful

Return_code
  EBUSY Aggregate could not be quiesced
  EINTR ZFS is shutting down
  EMVSERR Internal error using an osi service
  ENOENT Aggregate is not attached
  EPERM Permission denied to perform request

Reason_code
  0xEFnnxxxx See z/OS Distributed File Service Messages and Codes

266 z/OS V2R1.0 Distributed File Service zFS Administration
# Query Config Option

```c
short co_len; /* Length of structure */
char co_ver; /* Version of structure */
#define CO_VER_INITIAL 1 /* Initial version */
#define CO_SLEN 80 /* Size of string */
char co_string[CO_SLEN+1]; /* String value for option must be 0 terminated */
int co_value[4]; /* Place for integer values */
char co_reserved[24]; /* Reserved for future use */
} CFG_OPTION;

struct parmstruct { 
syscall_parmlist myparms;
CFG_OPTION co;
char system[9];
} myparmstruct;

int main(int argc, char **argv) {
int bpxrv;
int bpxrc;
int bpxrs;

CFG_OPTION *coptr = &(myparmstruct.co);
/* This next field should only be set if parms[1] is non-zero */
/* strcpy(myparmstruct.system,"DCEIMGVN"); */ /* set system to query */

myparmstruct.myparms.opcode = CFGOP_QUERY_ADM_THREADS;

myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);

myparmstruct.myparms.parms[1] = 0;
/* Only specify a non-zero offset for the next field (parms[1]) if you are */
/* z/OS 1.7 and above, and you want to configquery to a different system */
/* myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(CFG_OPTION); */

myparmstruct.myparms.parms[2] = 0;
myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(coptr,0,sizeof(CFG_OPTION));
memcpy(coptr->co_eye,CFGO_EYE,4);

if (bpxrv < 0) {
printf("Error querying config -adm_threads, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
return bpxrc;
}
else {
printf("Config query -adm_threads = %s\n",myparmstruct.co.co_string);
return 0;
}
```

**Example 2:** The following example shows an API to query the syslevel.

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
#include <string.h>

#define ZFSCALL_CONFIG 0x40000006
#define CFGOP_QUERY_SYSLEVEL 238 /* Query config option - syslevel */
#define NO_SYSPLEX_SUPPORT 0 /* Not in a sysplex shared file system environment */
#define SYSPLEX_ADMIN_LEVEL 1 /* Admin level sysplex shared file system environment */
#define SYSPLEX_FILE_LEVEL 2 /* File level sysplex shared file system environment */
#define SYSPLEX_FILESYS_LEVEL 3 /* Sysplex-aware on a file system basis */

typedef struct syscall_parmlist_t { 
int opcode; /* Operation code to perform */
int parms[7]; /* Specific to type of operation, */
/* provides access to the parms */
/* parms[4]-parms[6] are currently unused */
} syscall_parmlist;

typedef struct config_option_t { 
char co_eye[4]; /* Eye catcher */
#define CFGO_EYE "CFGP"
short co_len; /* Length of structure */
```
Query Config Option

```c
char co_ver; /* Version of structure */
#define CO_VER_INITIAL 1 /* Initial version */
#define CO_SLEN 80 /* Sizeof string */
char co_string[CO_SLEN+1]; /* String value for option must be 0 terminated */
int co_value[4]; /* Place for integer values */
char coReserved[24]; /* Reserved for future use */
} CFG_OPTION;

struct parmstruct {
    syscall_parmlist myparms;
    CFG_OPTION co;
    char system[9];
} myparmstruct;

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    CFG_OPTION *coptr = &(myparmstruct.co);
    int sysplex_level;
    /* strcpy(myparmstruct.system,"DCEIMGVN"); */ /* set system to query */
    myparmstruct.myparms.opcode = CFGOP_QUERY_SYSLEVEL;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = 0;
    /* myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(CFG_OPTION); */
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;
    memset(coptr,0,sizeof(CFG_OPTION));
    memcpy(coptr->co_eye,CFGO_EYE,4);
    coptr->co_ver=CO_VER_INITIAL;
    coptr->co_len=(int) sizeof(CFG_OPTION);

    BPIX1PCT("ZFS ",
        ZFSFILE_CONFIG, /* Config operation */
        sizeof(myparmstruct), /* Length of Argument */
        (char *) &myparmstruct, /* Pointer to Argument */
        &bpxrv, /* Pointer to Return_value */
        &bpxrc, /* Pointer to Return_code */
        &bpxrs); /* Pointer to Reason_code */
    if (bpxrv < 0) {
        printf("Error querying config -syslevel, BPXRV = %d BPXRC = %d BPXRS = %x
",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    }
    else {
        /* Parse our configquery string */
        /* format is */
        /* Prior to R11 APAR OA29619 or when sysplex=filesys not specified */
        /* OSlevel
         *Servicelevel
         *created timestamp
         *sysplex_state
         *interface_level
         */
        /* After R11 APAR OA29619 and when sysplex=filesys is specified */
        /* OSlevel
         *create timestamp
         *sysplex_state
         *interface_level
         *rwshare_default
         */
        version = myparmstruct.co.co_string;
        service = strchr(version, '\n'); /* find the end of the version (for 2nd line) */
        service++;
        /* ensure end of string for version string */
        created = strchr(service, '\n'); /* find the end of the service (for 2nd line) */
        created++;
        /* increment to next field (service) */
        sysplex = strchr(created, '\n'); /* find the end of the creation timestamp */
        sysplex++;
        /* increment to next field (creation) */
        interface = strchr(sysplex, '\n'); /* find end of the sysplex_state */
        interface++;
        /* increment to next field (interface level) */
        sysplex_level = atoi(sysplex);
        if (sysplex_level == NO_SYSPLEX_SUPPORT)
        {
            printf("zFS kernel: z/OS File System\nVersion %s Service Level %s.\nCreated on %s.\n",version,service,created);
        }
        else
        {
            char buffer[80];
        }
    }
}```
Quiesce Aggregate

Purpose

This subcommand call is an aggregate operation that quiesces a compatibility mode aggregate. It quiesces activity on the aggregate and its file system.

Format

<table>
<thead>
<tr>
<th>syscall_parmlist</th>
<th>opcode 132 AGOP QUIESCE_PARMDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>_parms[0]</td>
<td>offset to AGGR_ID</td>
</tr>
<tr>
<td>_parms[1]</td>
<td>offset to handle returned by quiesce</td>
</tr>
<tr>
<td>_parms[2]</td>
<td>0</td>
</tr>
<tr>
<td>_parms[3]</td>
<td>0</td>
</tr>
<tr>
<td>_parms[4]</td>
<td>0</td>
</tr>
<tr>
<td>_parms[5]</td>
<td>0</td>
</tr>
<tr>
<td>_parms[6]</td>
<td>0</td>
</tr>
</tbody>
</table>

AGGR_ID

| aid_eye          | char[6] "AGID" |
| aid_len          | char sizeof(AGGR_ID) |
| aid_ver          | char 1 |
| aid_name         | char[45] "OMVS.PRV.AGGR001.LDS0001" |
| aid_reserved     | char[33] 0 |
| quiesce_handle   | long |

Return_value 0 if request is successful, -1 if it is not successful

Usage notes

- Quiesce Aggregate is used to suspend activity on an aggregate. All activity on the file system contained in the aggregate that is mounted is also suspended. This subcommand is typically used before backing up an aggregate. The aggregate must be attached to be quiesced. The quiesce operation returns a quiesce handle that must be supplied on the unquiesce call.
Quiesce Aggregate

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

- Unquiesce Aggregate

Restrictions

None.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>
#define ZFSCALL_AGGR 0x40000005
#define AGOP_QUIESCE_PARMDATA 132

typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44
typedef struct aggr_id_t {
    char aid_eye[4]; /* Eye catcher */
#define AID_EYE "AGID"
    char aid_len; /* Length of this structure */
    char aid_ver; /* Version */
#define AID_VER_INITIAL 1 /* Initial version */
    char aid_name[ZFS_MAX_AGGRNAME+1]; /* Name, null terminated */
    char aid_reserved[33]; /* Reserved for the future */
} AGGR_ID;

struct parmstruct {
    syscall_parmlist myparms;
    AGGR_ID aggr_id;
    long quiesce_handle;
} ;

int main(int argc, char **argv) {
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";
    long save_quiesce_handle;
    struct parmstruct myparmstruct;
    AGGR_ID *idp = &(myparmstruct.aggr_id);
    myparmstruct.nyparms.opcode = AGOP_QUIESCE_PARMDATA;
    myparmstruct.nyparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.nyparms.parms[1] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
    myparmstruct.nyparms.parms[2] = 0;
    myparmstruct.nyparms.parms[3] = 0;
    myparmstruct.nyparms.parms[4] = 0;
    myparmstruct.nyparms.parms[5] = 0;
    myparmstruct.nyparms.parms[6] = 0;
    memset(&myparmstruct.aggr_id,0,sizeof(AGGR_ID)); /* Ensure reserved fields are 0 */
    memcpy(&myparmstruct.aggr_id.AID_EYE,4);
    myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
    myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
    strncpy(myparmstruct.aggr_id.aid_name,aggrname);
    BPX1PCT("ZFS",
    ZFSCALL_AGGR, /* Aggregate operation */
    sizeof(myparmstruct), /* Length of Argument */
    (char *) &myparmstruct, /* Pointer to Argument */
    Quiesce Aggregate
    270
    z/OS V2R1.0 Distributed File Service zFS Administration

```
Reset Backup Flag

Purpose

This subcommand call is used by backup programs to reset the backup bit after completion of a backup. The backup program is expected to quiesce the aggregate and save the quiesce handle before beginning the backup. After completing the backup, the backup bit should be reset before unquiescing the aggregate.

Format

```
syscall_parmlist
  opcode 157 AGOP_RESETFLAG_PARMDATA
  parms[0] offset to AGGR_ID
  parms[1] quiesce handle
  parms[2] 0
  parms[3] 0
  parms[4] 0
  parms[5] 0
  parms[6] 0

  AGGR_ID
    aid.eye char[4] "AGID"
    aid.len char sizeof(AGGR_ID)
    aid.ver char 1
    aid.name char[45] "OMVS.PRIV.AGGR001.LDS0001"
    aid_reserved char[33] 0

  Return_value 0 if request is successful, -1 if it is not successful

  Return_code
    EINVAL Invalid input parameters
    ENOENT Aggregate not found
    ENOSYS Aggregate not locally owned
    EBUSY Aggregate is growing
    ENOSERR Internal error using an osi service

  Reason_code
    OxEFnnxxxx See z/OS Distributed File Service Messages and Codes
```

Usage notes

- The backup bit must be reset while the aggregate is still quiesced for backup.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.
Reset Backup Flag

Related services
Quiesce Aggregate
Unquiesce Aggregate

Restrictions
None.

Examples

```c
#include <stdio.h>
#define ZFSCALL_AGGR 0x40000005
#define AGOP_RESETFLAG_PARAMDATA 105

typedef struct syscall_parmlist_t
{
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[2]-parms[6] are currently unused*/
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t
{
    char aid_eye[4]; /* Eye Catcher */
    #define AID_EYE "AGID"
    char aid_len; /* Length of this structure */
    char aid_ver; /* Version */
    #define AID_VER_INITIAL 1 /* Initial version */
    char aid_name[ZFS_MAX_AGGRNAME+1]; /* aggr name, null terminated */
    char aid_reserved[33]; /* Reserved for the future */
} AGGR_ID;

struct parmstruct
{
    syscall_parmlist myparms;
    AGGR_ID aggr_id;
};

int main(int argc, char
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char aggrname[45] = "PLEX.JMS.AGGR001.LDS0001"; /* aggregate name to attach */
    struct parmstruct myparmstruct;
    AGGR_ID *idp = &(myparmstruct.aggr_id);
    int quiesce_handle; /* This is the handle returned by zFS on a quiesce aggregate */

    myparmstruct.myparms.opcode = AGOP_RESETFLAG_PARAMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = quiesce_handle;
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    memset(idp,0,sizeof(AGGR_ID)); /* Ensure reserved fields are 0 */
    memcpy(&myparmstruct.aggr_id.aid_eye,AID_EYE,4);
    myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
    myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
    strcpy(myparmstruct.aggr_id.aid_name,aggrname);

    BPX1PCT("ZFS ",
        ZFSCALL_AGGR, /* Aggregate operation */
        sizeof(myparmstruct), /* Length of Argument */
        (char *) &myparmstruct, /* Pointer to Argument */
        &bpxrv, /* Pointer to Return_value */
        &bpxrc, /* Pointer to Return_code */
        &bpxrs); /* Pointer to Reason_code */

    if (bpxrv < 0)
    {
        printf("Error resetting backup flag for aggregate %s\n", aggrname);
        printf("BPRV = %d BPRRC = %d BPRXS = %lx",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    } else /* Return from attach was successful */
    {
```
Reset Backup Flag

print("Successfully reset backup flag for aggregate \%s\",aggrname);
}
return 0;
}

Set Auditfid

Purpose

This subcommand is an aggregate operation that sets the current value of the auditfid. The aggregate whose auditfid is to be changed must be attached.

Format

```c
syscall_parmlist

opcode 149    AGOP_SETAUDITFID_PARMDATA
parms[0] offset to AGGR_ID
parms[1]
0= set new auditfid if current auditfid is 0
1= set new auditfid regardless of current value (force)
2= set new auditfid to 0 (old)
parms[2]
0
parms[3]
0
parms[4]
0
parms[5]
0
parms[6]
0
AGGR_ID

aidEye char[4] "AGID"
aid_len char sizeof(AGGR_ID)
aid_ver char 1
aid_name char[45] "OMVS.PRV.AGGR001.LDS0001"
aid_reserved char[33] 0

return_value 0 if request is successful, -1 if it is not successful

return_code

EBUSY auditfid could not be set
EINVAL ZFS is shutting down
ENMSERR Internal error using an osi service
ENOTENT Aggregate is not attached
EPERM Permission denied to perform request

reason_code

0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must be UID=0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

List Aggregate Status (Version 2)

Restrictions

The aggregate cannot be attached as read-only. The aggregate cannot be quiesced. The aggregate cannot be in the process of being moved by zFS.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_SETAUDITFID_PARMDATA 149 /* Set or reset auditfid */
```

Chapter 13. zFS application programming interfaces  273
Set Auditfid

typedef struct syscall_parmlist_t {
   int opcode; /* Operation code to perform */
   int parms[7]; /* Specific to type of operation, */
   /* provides access to the parms */
   /* parms[4]-parms[6] are currently unused */
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t {
   char aid_eye[4]; /* Eye catcher */
   char aid_len; /* Length of this structure */
   char aid_ver; /* Version */
   char aid_name[ZFS_MAX_AGGRNAME+1]; /* Name, null terminated */
   char aid_reserved[33]; /* Reserved for the future */
} AGGR_ID;

struct parmstruct {
   syscall_parmlist myparms;
   AGGR_ID aggr_id;
};

int main(int argc, char **argv) {
   int bpxrv;
   int bpxrc;
   int bpxrs;
   char aggrname[45] = "OMVS.PRIV.AGGR001.LDS0001"; /* aggregate name to set auditfid */

   struct parmstruct myparmstruct;
   AGGR_ID *idp = &(myparmstruct.aggr_id);

   myparmstruct.myparms.opcode = AGOP_SETAUDITFID_PARM_DATA;
   myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
   myparmstruct.myparms.parms[1] = 0; /* 0=set new auditfid if current auditfid is 0 */
   myparmstruct.myparms.parms[2] = 0;
   myparmstruct.myparms.parms[3] = 0;
   myparmstruct.myparms.parms[4] = 0;
   myparmstruct.myparms.parms[5] = 0;
   myparmstruct.myparms.parms[6] = 0;

   memset(&myparmstruct.aggr_id,0,sizeof(AGGR_ID)); /* Ensure reserved fields are 0 */
   memcpy(&myparmstruct.aggr_id,AID_EYE,4);
   myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
   myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
   strcpy(myparmstruct.aggr_id.aid_name,aggrname);

   if (bpxrv < 0) {
      printf("Error setting auditfid for aggregate %s\n", aggrname);
      return bpxrc;
   } else /* Return from set auditfid was successful */ {
      printf("Aggregate %s set auditfid successfully\n",aggrname);
   }
   return 0;
}

Set Config Option

Purpose

Set Config Option is a set of subcommand calls (that are configuration operations) that set the current value for a particular configuration setting. Each one sets the configuration setting from input specified as a character string.
Set Config Option

The following Format and Example use the CFGOP_ADM_THREADS subcommand. The other set subcommands (see Table 10 on page 209) operate similarly. That is, each sets the configuration setting from the character string in the co_string field.

**Format**

```c
syscall_parmlist
  opcode 150  CFGOP_ADM_THREADS
  parms[0] offset to CFG_OPTION
  parms[1] offset to system name (optional)
  parms[2] 0
  parms[3] 0
  parms[4] 0
  parms[5] 0
  parms[6] 0

CFG_OPTION
  co_eye char[4] "CFOP"
  co_len short sizeof(CFG_OPTION)
  co_ver char 1
  co_string char[81] "15" /* New value for adm_threads */
  co_value_reserved int 0
  co_reserved char[24] 0
  systemname char[9]

Return_value  0 if request is successful, -1 if it is not successful

Return_code
  EBUSY Aggregate could not be quiesced
  EINTR ZFS is shutting down
  EMVSERR Internal error using an osi service
  ENOENT Aggregate is not attached
  EPERM Permission denied to perform request

Reason_code
  0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
```

**Usage notes**

- Reserved fields and undefined flags must be set to binary zeros.

**Privilege required**

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

**Related services**

- Query Config Option

**Restrictions**

None.

**Examples**

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include "stdio.h"
#define ZFSCALL_CONFIG 0x40000006
#define CFGOP_ADM_THREADS 150 /* Set number of admin threads */

typedef struct syscall_parmlist_t {
  int opcode;  /* Operation code to perform */
```
### Set Config Option

```c
int parms[7]; /* Specific to type of operation, */
/* provides access to the parms */
/* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct config_option_t {
    char co_eye[4]; /* Eye catcher */
    #define CFGO_EYE "CFOP"
    short co_len; /* Length of structure */
    char co_ver; /* Version of structure */
    #define CO_VER_INITIAL 1 /* Initial version */
    #define CO_SLEN 80 /* Sizeof string */
    char co_string[CO_SLEN+1]; /* String value for option must be 0 terminated */
    int co_value[4]; /* Place for integer values */
    char co_reserved[24]; /* Reserved for future use */
} CFG_OPTION;

struct parmstruct {
    syscall_parmlist myparms;
    CFG_OPTION co;
    char system[9];
} myparmstruct;

char new_adm_threads[CO_SLEN+1]="20"; /* New adm_threads value */

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;

    CFG_OPTION *coptr = &(myparmstruct.co);
    /* This next field should only be set if parms[1] is non-zero */
    /* strcpy(myparmstruct.system,"DCEIMGVN"); /* set system to change */

    myparmstruct.myparms.opcode = CFGOP_ADM_THREADS;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = 0;
    /* Only specify a non-zero offset for the next field (parms[1]) if */
    /* you are running z/OS 1.7 and above, and */
    /* you want to configquery to a different system */
    /* myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(CFG_OPTION); */

    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    memset(coptr,0,sizeof(CFG_OPTION));
    memcpy(coptr->co_eye,CFGO_EYE,4);
    coptr->co_ver=CO_VER_INITIAL;
    coptr->co_len=(int) sizeof(CFG_OPTION);
    strcpy(coptr->co_string,new_adm_threads); /* set new adm_thread value */

    BPX1PCT("ZFS ",
            ZFS_CALL_CONFIG, /* Config operation */
            sizeof(myparmstruct), /* Length of Argument */
            (char *) &myparmstruct, /* Pointer to Argument */
            &bpxrv, /* Pointer to Return_value */
            &bpxrc, /* Pointer to Return_code */
            &bpxrs); /* Pointer to Reason_code */

    if (bpxrv < 0) {
        printf("Error setting config -admThreads, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    } else {
        printf("Config -adm_threads = %s\n",myparmstruct.co.co_string);
    }
    return 0;
}
```

### Statistics Directory Cache Information

#### Purpose

This subcommand returns directory cache counters, including the number of requests, hits and discards from the directory cache.

**Note:** As of z/OS V1R13, this subcommand is no longer used. All output from a call to statistics directory cache information will be zeros.
Statistics Directory Cache Information

Format

```c
syscall_parmlist

opcode 249  STATOP_DIR_CACHE
parms[0] offset to STAT_API
parms[1] offset to output buffer
parms[2] offset to system name (optional)
parms[3] 0
parms[4] 0
parms[5] 0
parms[6] 0

STAT_API
sa_eye    char[4]   "STAP"
sa_len    int       length of buffer that follows STAT_API
sa_ver    int       1
sa_flags  char[1]   0x00
SA_RESET  0x00      Reset statistics
sa_fill   char[3]   0
sa_reserve int[4]   0
posix_time_high unsigned long high order 32 bits since epoch
posix_time_low unsigned long low order 32 bits since epoch
posix_useconds unsigned long microseconds

pad1 int

API_DIR_STATS
ad_eye    char[4]   "ADIR"
ad_size   short     size of output
ad_version char       version
ad_reserved1 char      reserved byte
ad_reserved int       always zero
ad_buffers int       number of buffers in the cache
ad_buffersize int     size of each buffer in K bytes
ad_resl    int       reserved
ad_requests int       requests to the cache
ad_reserved int       reserved
ad_hits    int       hits in the cache
ad_reserved int       reserved
ad_discards int       discards of data from the cache
ad_reserved2 int[10] reserved

systemname char[9]
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

Statistics Vnode Cache Information
Statistics Metadata Cache Information

Restrictions

None.

Examples

```c
#include <stdlib.h>
#include <stdio.h>

#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
/* #include <stdlib.h> */ /*
#include <stdio.h>
*/
```

Chapter 13. zFS application programming interfaces  277
Statistics Directory Cache Information

```c
#define ZFSCALL_STATS 0x40000007
#define STATOP_DIR_CACHE 249 /* Directory cache stats */
#define u_long unsigned long
#define CONVERT_RATIO_TO_INTS(RATIO, INTEGER, DECIMAL) {
    INTEGER = (int)RATIO;
    DECIMAL = (int)((RATIO - (double)INTEGER) * (double)1000.0);
}

typedef struct syscall_parmlist_t {
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation. */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused */
} syscall_parmlist;

typedef struct hyper {
    unsigned long high; /* unsigned long reserved */
    unsigned long low;
} hyper;

typedef struct API_DIR_STATS_t {
    char ad_eye[4]; /* Eye catcher = ADIR */
    short ad_size; /* Size of output structure */
    char ad_version; /* Version of stats */
    char ad_reserved1; /* Reserved byte, 0 in version 1 */
    hyper ad_buffers; /* Number of buffers in cache */
    int ad_buffsize; /* Size of each buffer in K bytes */
    int ad_res1; /* Reserved for future use, zero in version 1 */
    hyper ad_requests; /* Requests to the cache */
    hyper ad_hits; /* Hits in the cache */
    hyper ad_discards; /* Discards of data from cache */
    int ad_reserved[10]; /* Reserved for future use */
} API_DIR_STATS;

/* reset timestamp */
typedef struct reset_time {
    u_long posix_time_high; /* high order 32 bits since epoch */
    u_long posix_time_low; /* low order 32 bits since epoch */
    int posix_usecs; /* microseconds */
    int pad1;
} RESET_TIME;

/* The following structure is the api query control block */
typedef struct stat_api_t {
    char sa_eye[4]; /* 4 byte identifier must be */
    int sa_len; /* length of the buffer to put data into */
    char sa_version; /* the version number currently always */
    int sa_reserve[4]; /* Reserved */
    struct reset_time reset_time_info;
} STAT_API;

struct parmstruct {
    syscall_parmlist myparms;
    STAT_API myapi;
    API_DIR_STATS mystats;
    char systemname[9];
} myparmstruct;

int main(int argc, char **argv) {
    int bpxrv;
    int bpxrc;
    int bpxrs;
    int i;
    double temp_ratio;
    int whole, decimal;
    STAT_API *stapptr = &myparmstruct.myapi;
    char buf[33];
    myparmstruct.myparms.opcode = STATOP_DIR_CACHE;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(myparmstruct.myapi);
    myparmstruct.myparms.parms[2] = 0;
    
    // ... (rest of the code) ...
}```
Statistics Directory Cache Information

/ * Only specify a non-zero offset for the next field (parms[2]) if */
/ * you are running z/OS 1.7 and above, and */
/ * you want to query the directory cache statistics of a different system than this one */

myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) + sizeof(API_DIR_STATS);

myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(stapptr,0,sizeof(STAT_API));
memcpy(stapptr->sa_eye,SA_EYE,4);
stapptr->sa_ver=SA_VER_INITIAL;
stapptr->sa_len=(int) sizeof(API_DIR_STATS);

/ * This next field should only be set if parms[2] is non-zero */
/ * strcpy(myparmstruct.systemname,"DCEIMGVQ"); */

BPX1PCT("ZFS",
ZFSCALL_STATS, /* Perf statistics operation */
sizeof(myparmstruct), /* Length of Argument */
(char *) &myparmstruct, /* Pointer to Argument */
&bpxrv, /* Pointer to Return_value */
&bpxrc, /* Pointer to Return_code */
&bpxrs); /* Pointer to Reason_code */

if( bpxrv<0)
{
    printf("Error querying directory cache, BPRV = %d BPXRC = %d BPXRS = %x",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else
{
    printf("Directory Backing Caching Statistics\n");
    printf("Buffers (K bytes) Requests Hits Ratio Discards\n");
    printf("---------- --------- ---------- ---------- ------ ----------\n");

    temp_ratio = (myparmstruct.mystats.ad_requests.low == 0) ? 0.0 :
        ((double)myparmstruct.mystats.ad_hits.low)/myparmstruct.mystats.ad_requests.low;
    temp_ratio *= 100.0;
    CONVERT_RATIO_TO_INTS(temp_ratio,whole, decimal);
    decimal = decimal / 100; /* Just want tenths */
    printf("%10u %9u %10u %10u %3u.%1.1u\n", myparmstruct.mystats.ad_buffers.low,
        myparmstruct.mystats.ad_buffers.low * myparmstruct.mystats.ad_buffsize, myparmstruct.mystats.ad_requests.low,
        myparmstruct.mystats.ad_hits.low,
        whole, decimal);

    if (0==ctime_r((time_t *) &stapptr->reset_time_info.posix_time_low, buf))
    {
        printf("Could not get timestamp\n");
    }
    else
    {
        /* Insert the microseconds into the displayable time value */
        if (0==ctime_r((time_t *) &stapptr->reset_time_info.posix_time_low, buf))
        {
            printf("Could not get timestamp\n");
        }
    }
}
return 0;

Statistics iobyaggr Information

Purpose

This subcommand displays information about the number of reads and writes (I/Os) and the amount of data (in bytes) transferred for each aggregate.
Statistics iobyagar Information

Format

```
syscall_parmlist
opcode            244          STATOP_IOPYAGGR
parms[0]          offset to STAT_API
parms[1]          offset to output buffer
parms[2]          offset to system name (optional)
parms[3]          0
parms[4]          0
parms[5]          0
parms[6]          0
STAT_API
sa_eye            char[4]      "STAP*
sa_len            int           length of buffer that follows STAT_API
sa_flags          int           1
sa_RESET          0x00          Reset statistics
sa_fill           char[3]      0
sa_reserve        int[4]       0
posix_time_high   unsigned long high order 32 bits since epoch
posix_time_low    unsigned long low order 32 bits since epoch
posix_useconds    unsigned long microseconds
pad1              int
IO_REPORT2_GRAND_TOTALS

io_count          int          count of IO_REPORT2 lines
grand_total_reads unsigned long total reads
grand_total_writes unsigned long total writes
grand_total_read_bytes unsigned long total bytes read (in kilobytes)
grand_total_write_bytes unsigned long total bytes written (in kilobytes)
grand_total_devices unsigned long total number of aggregates
grand_total_waits_for_io unsigned long total number of waits for I/O
grand_total_waits_for_io_whole unsigned long average wait time (whole number)
grand_total_waits_for_io_decimal unsigned long average wait time in milliseconds
average_wait_time_for_io_decimal unsigned long average wait time in milliseconds
decimal part is in thousandths
3 means .003 and 300 means .3

IO_REPORT2[io_count]

volser             char[8]      DASD volser where aggregate resides
pavios             unsigned long max number of concurrent I/Os that zFS will issue
read_ind           char[4]      R/O or R/W (how aggregate is attached)
temp_read_bytes    unsigned long bytes read for this aggregate
temp_write_bytes   unsigned long bytes written for this aggregate (in kilobytes)
allocation_dsname  char[86]     data set name of aggregate
systemname         char[9]
```

Usage notes
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required
None.

Related services
- Statistics iobydasd Information
- Statistics iocounts Information

Restrictions
None.
Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>
#define ZFSCALL_STATS 0x40000007
#define STATOP_IOBYAGGR 244 /* Performance API queries */
#define E2BIG 145
#define u_long unsigned long
typedef struct syscall_parmlist_t
{
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused */
} syscall_parmlist;

typedef struct reset_time {
    u_long posix_time_high; /* high order 32 bits since epoch */
    u_long posix_time_low; /* low order 32 bits since epoch */
    u_long posix_usecs; /* microseconds */
    int pad1;
} RESET_TIME;

/*****************************************************************************************************/
/* The following structure is the api query control block */
/* It is used for all api query commands */
/*****************************************************************************************************/

typedef struct stat_api_t
{
    #define SA_EYE "STAP"
    char sa_eye[4]; /* 4 byte identifier must be */
    int sa_len; /* length of the buffer to put data into*/
    int sa_ver; /* the version number currently always 1*/
    #define SA_VER_INITIAL 0x01
    char sa_flags; /* flags field must be x00 or x80, x80 means reset statistics*/
    #define SA_RESET 0x80
    char sa_fill[3]; /* spare bytes */
    int sa_reserve[4]; /* Reserved */
    struct reset_time reset_time_info;
} STAT_API;

typedef struct io_report2_t
{
    char volser[8];
    unsigned long pavios;
    char read_ind[4];
    unsigned long temp_reads;
    unsigned long temp_read_bytes;
    unsigned long temp_writes;
    unsigned long temp_write_bytes;
    char allocation_dname[84];
} IO_REPORT2;

typedef struct io_report2_grand_totals_t
{
    int io_count; /* number IO_REPORT2 structs in buffer */
    unsigned long grand_total_reads; /* Total # reads */
    unsigned long grand_total_writes; /* Total # writes */
    unsigned long grand_total_read_bytes; /* Total bytes read */
    unsigned long grand_total_write_bytes; /* Total bytes written*/
    unsigned long grand_total_devices; /* Total # aggregates */
    unsigned long total_number_waits_for_io;
    unsigned long average_wait_time_for_io_whole; /* in milliseconds */
    unsigned long average_wait_time_for_io_decimal; /* in thousands */
    /* for example, 3 means .003 and 300 means .3 */
} IO_REPORT2_GRAND_TOTALS;

struct parmstruct
{
    syscall_parmlist myparms;
    STAT_API myapi;
    /* output buffer IO_REPORT2_GRAND_TOTALS + multiple IO_REPORT2s */
    char systemname[9];
} myparmstruct;

int main(int argc, char **argv)
{
    int bpvr; int bpvc; int bpxs; int i;
    IO_REPORT2_GRAND_TOTALS *stgt;
    IO_REPORT2 *str2;
    char *stsy;
}"

Chapter 13. zFS application programming interfaces 281
### Statistics iobyaggr Information

```c
char buf[33];
struct parmstruct *myp = &myparmstruct;
int mypsize, buflen;
STAT_API *stapptr = &(myparmstruct.myapi);

myparmstruct.myparms.opcode = STATOP_IOBYAGGR;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
myparmstruct.myparms.parms[2] = 0;

/* Only specify a non-zero offset for the next field (parms[2]) if */
/* you are running z/OS 1.7 and above, and */
/* you want to query the iobyaggr statistics of a different system than this one */

myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API);

myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;
memset(stapptr,0,sizeof(STAT_API));
memcpy(stapptr->sa_eye,SA_EYE,4);
stapptr->sa_ver=SA_VER_INITIAL;
stapptr->sa_len=0;

/* This next field should only be set if parms[2] is non-zero */
/* strcpy(myparmstruct.systemname,"DCEIMGVQ"); */
BPX1PCT("ZFS ",
  ZFSCALL_STATS, /* Perf statistics operation */
  sizeof(myparmstruct), /* Length of Argument */
  (char *) &myparmstruct, /* Pointer to Argument */
  &bpxrv, /* Pointer to Return_value */
  &bpxrc, /* Pointer to Return_code */
  &bpxrs); /* Pointer to Reason_code */

if( bpxrv < 0 )
{
  if( bpxrc == E2BIG )
  {
    buflen = stapptr->sa_len; /* Get buffer size needed */
    mypsize = sizeof(syscall_parmlist) + buflen +
    sizeof(myparmstruct.systemname);
    myp = (struct parmstruct *) malloc ((long) mypsize);
    memset(myp, 0, mypsize);
    printf("Need buffer size of %d, for a total of %d
", buflen,mypsize);

    stapptr->sa_len = buflen;
    strncpy(systemname,"DCEIMGVQ",20);
    stapptr->sa_len = sizeof(systemname);
    stapptr->sa_ver = SA_VER_INITIAL;
    stapptr = (STAT_API *)((char *)myp + sizeof(syscall_parmlist));
    memcpy(stapptr->sa_eye,SA_EYE,4);
    stapptr->sa_len = buflen;
    stgt = (IO_REPORT2_GRAND_TOTALS *)((char *)myp + sizeof(syscall_parmlist) +
    sizeof(STAT_API));
    str2 = (IO_REPORT2 *)((char *)stgt + sizeof(IO_REPORT2_GRAND_TOTALS));
    stsy = (char *)((char *)myp + sizeof(syscall_parmlist) +
    sizeof(STAT_API) + buflen);

    /* This next field should only be set if parms[2] is non-zero */
    /* strcat(stsy,"DCEIMGVQ"); */
    BPX1PCT("ZFS ",
      ZFSCALL_STATS, /* Aggregate operation */
      sizeof(syscall_parmlist) + buflen, /* Length of Argument */
      (char *) &myparmstruct, /* Pointer to Argument */
      &bpxrv, /* Pointer to Return_value */
      &bpxrc, /* Pointer to Return_code */
      &bpxrs); /* Pointer to Reason_code */

    if( bpxrv == 0 )
    {
      printf("ZFS I/O by Currently Attached Aggregate:
");
      printf("DASD PAV
");
      printf("VOLSER IOs Mode Reads K bytes Writes K bytes Dataset Name
");
      printf("---------- ---------- ----------- ----------- ---------------
");"
for( i = 0; i < stgt->io_count ; i++, str2++ )
{
    printf( "%6.6s %3u %10u %10u %10u %10u %-44.44s
", 
            str2->volser,
            str2->pavios,
            str2->read_ind,
            str2->temp_reads,
            str2->temp_read_bytes,
            str2->temp_write_bytes,
            str2->allocation_dsname);
}
printf( "%6u %10u %10u %10u %10u %-44.44s
", 
         stgt->grand_total_devices,
         stgt->grand_total_reads,
         stgt->grand_total_read_bytes,
         stgt->grand_total_writes,
         stgt->grand_total_write_bytes, "*TOTALS*" );
printf("\n");
printf("Total number of waits for I/O: %10u\n");
printf( "Average I/O wait time: %9u.%3.3u (msecs)\n", 
        stgt->average_wait_time_for_io_whole,
        stgt->average_wait_time_for_io_decimal );
printf("\n");
if (0==ctime_r((time_t *) &stapptr->reset_time_infoposix_time_low, buf))
{
    printf("Could not get timestamp.\n");
}
else /* Insert the microseconds into the displayable time value */
    {  /* Insert the microseconds into the displayable time value */
        strncpy(&buf[27],&buf[20],6);
        sprintf(&buf[20],"%06d",stapptr->reset_time_infoposix_usecs);
        buf[26]=' '; 
        buf[19]='.';
        printf("Last Reset Time: %s",buf);
    }
free(myp);
else /* iobyaggr failed with large enough buffer */
    {  /* iobyaggr failed with large enough buffer */
        printf("Error on iobyaggr with large enough buffer\n");
        printf("Error querying iobyaggr, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        free(myp);
        return bpxrc;
    }
else /* error was not E2BIG */
    {  /* Error on iobyaggr trying to get required size\n");
        printf("Error on iobyaggr trying to get required size\n");
        printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        free(myp);
        return bpxrc;
    }
else /* asking for buffer size gave rv = 0; maybe there is no data */
    {  /* asking for buffer size gave rv = 0; maybe there is no data */
        if( myparmstruct.myapi.sa_len == 0 )
        {  /* asking for buffer size gave rv = 0; maybe there is no data */
            printf("No data\n");
            printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        }
else /* No, there was some other problem with getting the size needed */
    {  /* No, there was some other problem with getting the size needed */
        printf("Error getting size required\n");
        printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    }
}
return 0;

Statistics iobyaggr Information

Purpose

This subcommand displays information about the number of reads and writes and
the number of bytes transferred for each DASD volume. The number of I/Os and
the amount of data transferred is determined on a DASD basis.
Statistics iobydasd Information

Format

```
syscall_parmlist
  opcode    245  STATOP_IOBYDASD
  parms[0]  offset to STAT_API
  parms[1]  offset to output buffer
  parms[2]  offset to system name (optional)
  parms[3]  0
  parms[4]  0
  parms[5]  0
  parms[6]  0

STAT_API
  sa_eye    char[4]  "STAP"
  sa_len    int     length of buffer that follows STAT_API
  sa_ver    int     1
  sa_flags  char[1]  0x00
    SA_RESET 0x00  Reset statistics
  sa_reserve int[4]  0
  posix_time_high unsigned long  high order 32 bits since epoch
  posix_time_low unsigned long   low order 32 bits since epoch
  posix_useconds unsigned long  microseconds
  pad1     int

API_IOBYDASD_HDR
  number_of_lines int  count of API_IOBYDASD_DATA lines
  pad        int  0
  grand_total_reserved int  always zero
  grand_total_waits hyper  total waits
  average_wait_time_whole int  average wait time (whole number)
  average_wait_time_decimal int  average wait time in milliseconds
    decimal part is in thousands
      3 means .003 and 300 means .3

API_IOBYDASD_DATA[number_of_lines]
  spare     int  0
  volser    char[6]  DASD volser
  filler    char[2]  reserved
  pavios    unsigned long  max number of concurrent I/Os zFS will issue for this DASD
  reads     unsigned long  count of reads for this DASD
  read_bytes unsigned long  bytes read for this DASD (in kilobytes)
  writes    unsigned long  count of writes for this DASD
  write_bytes unsigned long  bytes written for this DASD (in kilobytes)
  waits     unsigned long  waits
  avg_wait_whole int  average wait time (whole number)
  avg_wait_decimal int  average wait time in milliseconds
    decimal part is in thousands
      3 means .003 and 300 means .3
  systemname char[9]

Return_value  0 if request is successful, -1 if it is not successful

Return_code
  EINTR  zFS is shutting down
  EINVAL  Invalid parameter list
  EMVSERR  Internal error occurred
  E2BIG  Information too big for buffer supplied

Reason_code
  0xEFnnxxxx  See z/OS Distributed File Service Messages and Codes
```

Usage notes
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required
None.

Related services
- Statistics iobyaggr Information
- Statistics iocounts Information
Restrictions

None.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, char *, int *, int *, int *, int *);

#include <stdio.h>
#define ZFSCALL_STATS 0x40000007
#define STATOP_IOBYDASD 245 /* Performance API queries */
#define E2BIG 145
#define ENOMEM 132
#define u_long unsigned long

typedef struct syscall_parmlist_t
{
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
        /* provides access to the parms */
        /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct reset_time {
    u_long posix_time_high; /* high order 32 bits since epoch */
    u_long posix_time_low; /* low order 32 bits since epoch */
    u_long posix_usecs; /* microseconds */
    int pad;
} RESET_TIME;

typedef struct hyper_t {
    unsigned long high; /* unsigned long reserved */
    unsigned long low;
} hyper;

/*********************************************************************/
/* The following structure is the api query control block */
/* It is used for all api query commands */
𝕜 ipad; /* used for the buffer of the output of the API */

typedef struct stat_api_t
{
    #define SA_EYE "STAP"
    char sa_eye[4]; /* 4 byte identifier must be */
    int sa_len; /* length of the buffer to put data into*/
        /* this buffer area follows this struct*/
    int sa_ver; /* the version number currently always 1*/
    #define SA_VER_INITIAL 0x01
    char sa_flags; /* flags field must be x00 or x80, x80 means reset statistics*/
    #define SA_RESET 0x80
    char sa_fill[3]; /* spare bytes */
    int sa_reserve[4]; /* Reserved */
    struct reset_time reset_time_info;
} STAT_API;

typedef struct api_iobydasd_hdr
{
    int number_of_lines;
    int pad;
    hyper grand_total_waits;
    int avg_wait_time_wholed; /* in milliseconds */
    int avg_wait_time_decimal; /* in thousandths */
} API_IOBYDASD_HDR;

typedef struct api_iobydasd_data
{
    int spare;
    char volser[6];
    char filler[2];
    unsigned long pavios;
    unsigned long reads;
    unsigned long read_bytes;
    unsigned long writes;
    unsigned long write_bytes;
    unsigned long wait;
    int avg_wait_wholed;
    int avg_wait_decimal;
} API_IOBYDASD_DATA;

struct parmstruct
{
    syscall_parmlist myparms;
    STAT_API myapi;
    /* output buffer API_IOBYDASD_HDR + multiple API_IOBYDASD_DATAs */
}```
char systemname[9];
} myparmstruct;

int main(int argc, char *argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    int i;
    API_IOBYDASD_HDR *stdh;
    API_IOBYDASD_DATA *stdd;
    char *stsy;
    char buf[33];

    struct parmstruct *myp = &myparmstruct;
    int mypsize, buflen;

    STAT_API *stapptr = &(myparmstruct.myapi);

    myparmstruct.myparms.opcode = STATOP_IOBYDASD;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
    myparmstruct.myparms.parms[2] = 0;
    /* Only specify a non-zero offset for the next field (parms[2]) if */
    /* you are running z/OS 1.7 and above, and */
    /* you want to query the iobydasd statistics of a different system than this one */
    myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API); /*

    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;
    memset(stapptr,0,sizeof(STAT_API));
    memcpy(stapptr->sa_eye,SA_EYE,4);
    stapptr->sa_ver=SA_VER_INITIAL;
    stapptr->sa_len=0;
    /* This next field should only be set if parms[2] is non-zero */
    /* strcpy(myparmstruct.systemname,"DCEIMGVQ"); */
    BPX1PCT("ZFS", /* Perf statistics operation */
        ZFSCALL_STATS, /* Length of Argument */
        (char *) &myparmstruct, /* Pointer to Argument */
        &bpxrv, /* Pointer to Return_value */
        &bpxrc, /* Pointer to Return_code */
        &bpxrs); /* Pointer to Reason_code */

    if( bpxrv < 0 )
    {
        buflen = stapptr->sa_len; /* Get buffer size needed */
        mypsize = sizeof(myparmstruct) + sizeof(STAT_API) + buflen +
            sizeof(myparmstruct.systemname);
        myp = (struct parmstruct *) malloc ((long) mypsize);
        memset(myp, 0, mypsize);

        printf("Need buffer size of %d, for a total of %d\n", buflen, mypsize);
        myp->myparms.opcode = STATOP_IOBYDASD;
        myp->myparms.parms[0] = sizeof(syscall_parmlist);
        myp->myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
        myp->myparms.parms[2] = 0;
        /* Only specify a non-zero offset for the next field (parms[2]) if */
        /* you are running z/OS 1.7 and above, and */
        /* you want to query the iobydasd statistics of a different system than this one */
        myp->myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) + buflen; /*

        myp->myparms.parms[3] = 0;
        myp->myparms.parms[4] = 0;
        myp->myparms.parms[5] = 0;
        myp->myparms.parms[6] = 0;
        stapptr = (STAT_API *)((char *)myp + sizeof(syscall_parmlist));
        memcpy(stapptr->sa_eye,SA_EYE,4);
        stapptr->sa_ver=SA_VER_INITIAL;
        stapptr->sa_len=buflen;
        stdh = (API_IOBYDASD_HDR *)((char *)stapptr + sizeof(API_IOBYDASD_HDR));
        stdd = (API_IOBYDASD_DATA *)((char *)stdh + sizeof(API_IOBYDASD_HDR));
        stsy = (char *)((char *)myp + sizeof(syscall_parmlist) + sizeof(STAT_API) + buflen);
        /* This next field should only be set if parms[2] is non-zero */
        /* strcpy(stsy,"DCEIMGVQ"); */
    }
Statistics iobydasd Information

BPX1PCT("ZFS", /* ZFS stats operation */ 
ZFSCALL_STATS, /* Length of Argument */ 
mypsize, /* Pointer to Argument */ 
&bpxrv, /* Pointer to Return_value */ 
&bpxrc, /* Pointer to Return_code */ 
&bpxrs); /* Pointer to Reason_code */

if( bpxrv == 0 )
{
    printf("zFS I/O by Currently Attached DASD/VOLs\n");
    printf("\n");
    printf("DASD PAV\n");
    printf("VOLSER IOs Reads K bytes Writes K bytes Waits Average Wait\n");
    for( i=0; i< stdh->number_of_lines ; i++, stdd++ )
    {
        printf("%6.6s %3u %10u %10u %10u %10u %6u.%3.3u\n", 
            stdd->volser,
            stdd->pavios,
            stdd->reads,
            stdd->read_bytes,
            stdd->writes,
            stdd->write_bytes,
            stdd->waits,
            stdd->avg_wait_whole,
            stdd->avg_wait_decimal);
    }
    printf("\n");
    printf("Total number of waits for I/O: %u,,%u\n", 
        stdh->grand_total_waits.high,stdh->grand_total_waits.low);
    printf("Average I/O wait time: %9u.%3.3u (msecs)\n", 
        stdh->avg_wait_time_whole,
        stdh->avg_wait_time_decimal);
    printf("\n");
    if (0==ctime_r((time_t *) &stapptr->reset_time_info.posix_time_low, buf))
    {
        printf("Could not get timestamp.\n");
    }
    else
    { /* Insert the microseconds into the displayable time value */
        sprintf(&(buf[27]),"%06d",stapptr->reset_time_info.posix_usecs);
        buf[26]=' ';
        buf[19]='.';
        printf("Last Reset Time: %s",buf);
    }
    free(myp);
    else /* iobydasd failed with large enough buffer */
    {
        printf("Error on iobydasd with large enough buffer\n");
        printf("Error querying iobydasd, BPXRV = %d BPXRC = %d BPXRS = %x\n", 
            bpxrv,bpxrc,bpxrs);
        free(myp);
        return bpxrc;
    }
    else /* error was not E2BIG */
    {
        printf("Error on iobydasd trying to get required size\n");
        printf("BPXRV = %d BPXRC = %d BPXRS = %x,\n", 
            bpxrv,bpxrc,bpxrs);
        free(myp);
        return bpxrc;
    }
    else /* asking for buffer size gave rv = 0; maybe there is no data */
    { /*
        if( myparmstruct.myapi.sa_len == 0 )
        { /*
            printf("No data\n");
            printf("BPXRV = %d BPXRC = %d BPXRS = %x,\n", 
                bpxrv,bpxrc,bpxrs);
        }  /*
        else /* No, there was some other problem with getting the size needed */
        { /*
            printf("Error getting size required\n");
            printf("BPXRV = %d BPXRC = %d BPXRS = %x,\n", 
                bpxrv,bpxrc,bpxrs);
        }
    }
    return 0;
}
Statistics iocounts Information

Purpose

This subcommand displays information about how often zFS performs I/O for various circumstances and how often it waits on that I/O.

Format

```
syscall_parmlist
    opcode 243 STATOP_IOCOUNTS
    parms[0] offset to STAT_API
    parms[1] offset to output buffer
    parms[2] offset to system name (optional)
    parms[3] 0
    parms[4] 0
    parms[5] 0
    parms[6] 0

STAT_API
    sa_eye       char[4] "STAP"
    sa_len       int     length of buffer that follows STAT_API
    sa_ver       int     1
    sa_flags     char[1] 0x00
    SA_RESET     0x80     Reset statistics
    sa_fill char[3] 0
    sa_reserve int[4] 0
    posix_time_high unsigned long high order 32 bits since epoch
    posix_time_low unsigned long low order 32 bits since epoch
    posix_useconds unsigned long microseconds
    pad1 int

API_IO_BY_TYPE[3]
    number_of_lines unsigned long count of API_IO_BY_TYPE lines (3)
    count unsigned long count of I/Os for type
    waits unsigned long number of waits for type
    cancels unsigned long number of cancels for type
    merges unsigned long number of merges for type
    type reserved1 reserved
    description char[51] type description
    pad1 char[3] pad bytes

API_IO_BY_CIRC[19]
    number_of_lines unsigned long count of API_IO_BY_CIRC lines (19)
    count unsigned long count of I/Os for circumstance
    waits unsigned long number of waits for circumstance
    cancels unsigned long number of cancels for circumstance
    merges unsigned long number of merges for circumstance
    type reserved1 reserved
    description char[51] circumstance description
    pad1 char[3] pad bytes
    systemname char[9]
```

Usage notes

- The number of API_IO_BY_CIRC items can be 18 (for compatibility with previous releases) or 19.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.
Statistics iocounts Information

Related services
Statistics iobyaggr Information
Statistics iobysdasd Information

Restrictions
None.

Examples

```c
#define ZESCALL_STATS 0x40000007
#define STATOP_IOCOUNTS 243 /* Performance API queries */
#define TOTAL_TYPES 3
#define TOTAL_CIRC 19
#define u_long unsigned long

typedef struct syscall_parmlist_t 
{ 
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused */
} syscall_parmlist;

typedef struct reset_time 
{ 
    u_long posix_time_high; /* high order 32 bits since epoc */
    u_long posix_time_low; /* low order 32 bits since epoch */
    int posix_usecs; /* microseconds */
} RESET_TIME;

typedef struct stat_api_t 
{ 
    #define SA_EYE "STAP" 
    char sa_eye[4]; /* 4 byte identifier must be */
    int sa_len; /* length of the buffer to put data into */
    /* this buffer area follows this struct */
    int sa_ver; /* the version number currently always 1 */
    #define SA_VER_INITIAL 0x01
    char sa_fill[3]; /* spare bytes */
    char sa_reserve[4]; /* Reserved */
    struct reset_time reset_time_info;
} STAT_API;

typedef struct API_IO_BY_TYPE_t 
{ 
    unsigned long number_of_lines;
    unsigned long count;
    unsigned long waits;
    unsigned long cancels; /* Successful cancels of IO */
    unsigned long merges; /* Successful merges of IO */
    char reserved[6];
    char description[51];
    char pad[3];
} API_IO_BY_TYPE;

typedef struct API_IO_BY_CIRC_t 
{ 
    unsigned long number_of_lines;
    unsigned long count;
    unsigned long waits;
    unsigned long cancels;
    unsigned long merges;
    char reserved[6];
    char description[51];
    char pad[3];
} API_IO_BY_CIRC;
```

Statistics iocounts Information

Chapter 13. zFS application programming interfaces
struct parmstruct
{
    syscall_parmlist myparms;
    STAT_API myapi;
    API_IO_BY_TYPE mystatsbytype[TOTAL_TYPES];
    API_IO_BY_CIRC mystatsbycirc[TOTAL_CIRC];
    char systemname[9];
} myparmstruct;

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    int i;
    STAT_API *stapptr = &(myparmstruct.myapi);
    API_IO_BY_TYPE *stiotptr = &(myparmstruct.mystatsbytype[0]);
    API_IO_BY_CIRC *stiocptr = &(myparmstruct.mystatsbycirc[0]);
    char buf[33];
    myparmstruct.myparms.opcode = STATOP_IOCOUNCTS;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
    myparmstruct.myparms.parms[2] = 0;
    // Only specify a non-zero offset for the next field (parms[2]) if */
    // you are running z/OS 1.7 and above, and */
    // you want to query the iocounts of a different system than this one */
    myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) +
    // TOTAL_TYPES * sizeof(API_IO_BY_TYPE)) */
    // TOTAL_CIRC * sizeof(API_IO_BY_CIRC)); */
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;
    memset(stapptr,0,sizeof(STAT_API));
    memcpy(stapptr->sa_eye,SA_EYE,4);
    stapptr->sa_ver=SA_VER_INITIAL;
    stapptr->sa_len=(int) (TOTAL_TYPES * sizeof(API_IO_BY_TYPE)) +
    (TOTAL_CIRC * sizeof(API_IO_BY_CIRC));
    // This next field should only be set if parms[2] is non-zero */
    // strcpy(myparmstruct.systemname,"DCEIMGVQ"); */
    BPX1PCT("ZFS ",
        ZFSCALL_STATS,   /* Perf statistics operation */
        sizeof(myparmstruct), /* Length of Argument */
        (char *)&myparmstruct, /* Pointer to Argument */
        &bpxrv, /* Pointer to Return_value */
        &bpxrc, /* Pointer to Return_code */
        &bpxrs); /* Pointer to Reason_code */
    if( bpxrv < 0 )
    {
        printf("Error querying iocounts, BPXRV = %d BPXRC = %d BPXRS = %x
",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    }
    else
    {
        if( stiotptr->number_of_lines != TOTAL_TYPES )
        {
            printf("Unexpected number of IO Types, %d instead of TOTAL_TYPES\n",stiotptr->number_of_lines);
            return 1;
        }
        if( stiocptr->number_of_lines != TOTAL_CIRC )
        {
            printf("Unexpected number of IO Circumstances, %d instead of TOTAL_CIRC\n",stiocptr->number_of_lines);
            return 2;
        }
        printf(" 1/O Summary By Type\n");
        printf(" -------------------\n");
        printf("Count Waits Cancels Merges Type\n");
        printf("---------- ---------- ---------- ---------- ----------\n");
        for( i=0; i<TOTAL_TYPES; i++ )
        {
            printf("%10u %10u %10u %10u %s\n",stiotptr->count, stiotptr->waits, stiotptr->cancels, stiotptr->merges, stiotptr->description);
            stiotptr = stiotptr + 1;
        }
        printf(" \n");
        printf(" 1/O Summary By Circumstance\n");
        printf(" -------------------\n");
### Statistics iocounts Information

```c
printf("\n");
printf("Count Waits Cancels Merges Circumstance\n");
printf("---------- ---------- ---------- ---------- ------------\n");
for( i=0; i<TOTAL_CIRC; i++ )
{
    printf("%10u %10u %10u %10u %s\n",
            stiocptr->count, stiocptr->waits, stiocptr->cancels, stiocptr->merges,
            stiocptr->description);
    stiocptr = stiocptr +1;
    printf("\n");
}
```

```c
else
{/* Insert the microseconds into the displayable time value */
    sprintf(&buf[20],"%06d",stapptr->reset_time_info.posix_usecs);
    buf[26]=' '
    buf[19]='.'
    printf("Last Reset Time: %s",buf);
}
```  

### Statistics Kernel Information

#### Purpose

This subcommand call is a performance statistics operation that returns kernel counters, including the number of kernel operations and average time for the operation.
Statistics Kernel Information

Format

```
syscall_parmlist
  opcode  246    STATOP_KNPFS
  parms[0] offset to STAT_API
  parms[1] offset to output buffer
  parms[2] offset to system name (optional)
  parms[3] 0
  parms[4] 0
  parms[5] 0
  parms[6] 0

STAT_API
  sa_eye   char[4] "STAP"   length of buffer that follows STAT_API
  sa_len   int     length of output buffer
  sa_ver   int     1
  sa_res   int[4] 0
  sa_fill  char[3] 0
  posix_time_high unsigned long high order 32 bits since epoch
  posix_time_low unsigned long low order 32 bits since epoch
  posix_useconds unsigned long microseconds

KERNEL_CALL_STATS
  kc_eye   char[8] "xxxxADIR" operation name string
  kc_version short version
  kc_size   short size of output
  kc_totalops unsigned long grand total operations
  kc_totaltime int reserved
  kc_valid_slots int number of slots in above array that actually contains data
  kc_validslots int [10] reserved
  kc_reserved int [10] reserved
  kc_totaltime_reserved int reserved
  kc_totaltime int grand total wait time
  kc_time  two_words integer part of average time/fractional part of average time
  kc_valid char [6] reserved
  kc_count unsigned long count of operations
  kc_operation_name char [27] operation name string
  kc_valid int [10] reserved

KERNEL_LINE[40]
  kl_operation_name char[27] operation name string
  kl_count unsigned long count of operations
  kl_time  two_words integer part of average time/fractional part of average time
  kl_valid char[4] reserved
  kl_valid_slots int [10] reserved
  kl_reserved int reserved

systemname     char [9]
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

- Statistics Vnode Cache Information
- Statistics Metadata Cache Information

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

/* #include <stdlib.h> */
#include <stdio.h>
#define ZFSCALL_STATS 0x40000007
#define STATOP_KNPFS 246 /* Performance API queries */
#define u_long unsigned long
```

Return_value

0 if request is successful, -1 if it is not successful.

Return_code

- EINTR  zFS is shutting down
- EINVAL  Invalid parameter list
- EMVSERR  Internal error occurred
- E2BIG  Information too big for buffer supplied

Reason code

0xEFnnxxxx  See z/OS Distributed File Service Messages and Codes
typedef struct syscall_parmlist_t {
    int opcode;     /* Operation code to perform */
    int parms[7];   /* Specific to type of operation. */
                     /* Provides access to the parms */
                     /* parms[4]-parms[6] are currently unused */
} syscall_parmlist;

typedef union {
    struct double_word_t {
        unsigned int first_word;
        unsigned int second_word;
    } double_word;
    double alignment_dummy;
} two_words;

#define MAX_KERNEL_LINES 40

/* */
typedef struct KERNEL_line_t {
    char kl_operation_name[27];
    char kl_valid;
    u_long kl_count;
    two_words kl_time;
    int kl_reserved[6];
} KERNEL_LINE;

/* */
typedef struct kernel_call_stats_t {
    char kc_eye[8];   /* eye catcher */
    short kc_version;
    short kc_len;
    int pad1;
    KERNEL_LINE OUTPUT[MAX_KERNEL_LINES];
    u_long kc_totalops; /* Grand Total operations */
    int pad2;
    two_words kc_totaltime; /* Grand Total wait time */
    int kc_valid_slots; /* Number of slots in the above array */
    /* that actually contain data */
    int kc_reserved[10];
    int pad3;
} KERNEL_CALL_STATS;

/* reset timestamp */
typedef struct reset_time {
    u_long posix_time_high; /* high order 32 bits since epoc */
    u_long posix_time_low; /* low order 32 bits since epoch */
    u_long posix_usecs; /* microseconds */
    int pad1;
} RESET_TIME;

/******************************************************************************/
/* The following structure is the api query control block */
/* It is used for all api query commands */
/******************************************************************************/
typedef struct stat_api_t {
    #define SA_EYE "STAP"
    char sa_eye[4];     /* 4 byte identifier must be */
    int sa_len;         /* length of the buffer to put data into*/
                     /* this buffer area follows this struct */
    int sa_ver;         /* the version number currently always 1*/
    #define SA_VER_INITIAL 0x01
    char sa_flags;      /* flags field must be x00 or x80, x80 means reset statistics*/
    #define SA_RESET 0x80
    char sa_fill[3];    /* spare bytes */
    int sa_reserve[4];  /* Reserved */
    struct reset_time reset_time_info;
} STAT_API;

struct parmstruct {
    syscall_parmlist myparms;
    STAT_API myapi;
    KERNEL_CALL_STATS mystats;
    char systemname[9];
} myparmstruct;

int main(int argc, char **argv) {
    int bpxrv;
    int bpxrc;
    int bpxrs;
    int i;

    STAT_API *stapptr = &(myparmstruct.myapi);
    KERNEL_CALL_STATS *stkcptr = &(myparmstruct.mystats);
    char buf[33];

    myparmstruct.myparms.opcode = STATOP_KNPF;

    /* read in the kernel data */
Statistics Kernel Information

```
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
myparmstruct.myparms.parms[2] = 0;
/* Only specify a non-zero offset for the next field (parms[2]) if */
/* you are running z/OS 1.7 and above, and */
/* you want to query the kernel statistics of a different system than this one */
/* myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) + */
/* sizeof(KERNEL_CALL_STATS); */

myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;
memset(stapptr,0,sizeof(STAT_API));
memcpy(stapptr->sa_eye,SA_EYE,4);
stapptr->sa_ver=SA_VER_INITIAL;
stapptr->sa_len=(int) sizeof(KERNEL_CALL_STATS);
/* This next field should only be set if parms[2] is non-zero */
/* strcpy(myparmstruct.systemname,"DCEIMGVQ"); */

BPX1PCT("ZFS 
  ZFS_CALL_STATS, /* Perf statistics operation */
  sizeof(myparmstruct), /* Length of Argument */
  (char *) &myparmstruct, /* Pointer to Argument */
  &bpxrv, /* Pointer to Return_value */
  &bpxrc, /* Pointer to Return_code */
  &bpxrs); /* Pointer to Reason_code */
if( bpxrv<0 )
{
  printf("Error querying kernel calls, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
  return bpxrc;
}
else
{
  printf("%34s
","zFS Kernel PFS Calls");
  printf("%34s
","--------------------");
  printf("%34s\n","Operation Count Avg Time\n");
  printf("--------- ---------- ----------\n");
  i=0;
  while (myparmstruct.mystats.OUTPUT[i].kl_valid == 1)
  {
    printf("%13s %10u %9u.%3.3u\n",myparmstruct.mystats.OUTPUT[i].kl_operation_name,
           myparmstruct.mystats.OUTPUT[i].kl_count,
           myparmstruct.mystats.OUTPUT[i].kl_time.double_word.first_word,
           myparmstruct.mystats.OUTPUT[i].kl_time.double_word.second_word);
    i+=1;
  }
  printf("--------- ---------- ----------\n");
  printf("%34s\n","TOTALS\n");
  printf("%34s\n","%10u %9u.%3.3u\n",myparmstruct.mystats.kc_totalops,
           myparmstruct.mystats.kc_totaltime.double_word.first_word,
           myparmstruct.mystats.kc_totaltime.double_word.second_word);
  if (0!=ctime_r((time_t *) &stapptr->reset_time_info.posix_time_low, buf))
  {
    printf("Could not get timestamp.\n");
  }
  else
  { /* Insert the microseconds into the displayable time value */
    strncpy(&(buf[27]),&(buf[20]),6);
    sprintf(&(buf[20]),"%06d",stapptr->reset_time_info.posix_usecs);
    buf[26]= ' ';
    buf[19]= ' ';
    printf("Last Reset Time: %s",buf);
  }
  return 0;
}
```

Statistics Locking Information

Purpose

This subcommand call is a performance statistics operation that returns locking information.
Statistics Locking Information

Format

```
syscall_parmlist
  opcode 240 STATOP_LOCKING
  parm[0] offset to STAT_API
  parm[1] offset to STAT_LOCKING
  parm[2] offset to system name (optional)
  parm[3] 0
  parm[4] 0
  parm[5] 0
  parm[6] 0

STAT_API
  sa_eye char[4] "STAP"
  sa_len int sizeof(STAT_LOCKING)
  sa_yer int 1
  sa_flags char 0x80 for reset; 0x00 otherwise
  sa_fill char[3] 0
  sa_reserve int[4] 0

STAT_LOCKING
  reserved1 int
  stk_un timed_sleeps unsigned long number of untimed sleeps
  stk_timed_sleeps unsigned long number of timed sleeps
  stk_wakeups unsigned long number of wake ups
  stk_total_wait_for_locks unsigned long total waits for locks
  stk_average_lock_wait_time double average lock wait time
  stk_avg_lock_wait_time whole int average lock wait time in msecs left
  of the decimal part
  stk_avg_lock_wait_time_decimal int average lock wait time in msecs decimal portion
  decimal part is in thousands
  stk_total_monitored_sleeps unsigned long total monitored sleeps
  stk_average_monitored_sleep_time double average monitored sleep time
  stk_avg_mon_sleep_time whole int average monitored sleep time in msecs left of the
  decimal part
  stk_avg_mon_sleep_time_decimal int average monitored sleep time in msecs
  decimal portion
  decimal part is in thousands
  stk_total_contentions unsigned long total lock contention of all kinds
  stk_reserved_space char[48] reserved for future use
  stk_locks struct Lock_line[15] storage for the lock data
  count int Number of thread waits for this lock
  async int Asynchronous disposition
  spins int Number of attempts to get lock that did not
  resolve immediately
  percentage double
  percentage_whole int percentage >= 1
  percentage_decimal int decimal part is in thousands
  3 means .003 and 300 means .3
  description char[84] Description of the lock
  stk_sleeps struct Sleep_line[5] storage for sleep data
  sleepcount unsigned long Time spent sleeping
  percentage double Percentage of time spent sleeping
  percentage_whole int Percentage >= 1
  percentage_decimal int Percentage < 1
  description char[84] Description of the thread
  systemname char[9]
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.
Statistics Locking Information

Related services
Statistics Storage Information
Statistics User Cache Information

Restrictions
None.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

/* #include <stdlib.h> */
#include <stdio.h>
#define ZFSCALL_STATS 0x40000007
#define STATOP_LOCKING 240 /* Performance API queries */
#define u_long unsigned long

typedef struct syscall_parmlist_t
{
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation. */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist_t;

typedef struct Lock_line_t
{
    int count; /* Number of thread waits for this lock */
    int async; /* Asynchronous disposition*/
    int spins; /* Number of attempts to get lock that did not resolve immediately*/
    int pad;
    double percentage; /*%*/
    int percentage_whole; /* percentage >= 1*/
    int percentage_decimal; /* percentage < 1*/
    /* in thousands.*/
    /* For example, 3 means .003 and 300 means .3 */
    char description[84]; /* Description of the lock */
    int pad2;
} LOCK_LINE;

typedef struct Sleep_line_t
{
    unsigned long sleepcount; /* Time spent sleeping */
    int pad;
    double percentage; /* Percentage of time spent sleeping*/
    int percentage_whole; /* Percentage >=1 */
    int percentage_decimal; /* Percentage < 1*/
    /* in thousands.*/
    /* For example, 3 means .003 and 300 means .3 */
    char description[84]; /* Description of the thread */
    int pad2;
} SLEEP_LINE;

/* reset timestamp */
typedef struct reset_time {
    u_long posix_time_high; /* high order 32 bits since epoch */
    u_long posix_time_low; /* low order 32 bits since epoch */
    u_long posix_usecs; /* microseconds */
    int pad1;
} RESET_TIME;

/* The following structure is the api query control block */
typedef struct stat_api_t {
    #define SA_EYE "STAP"
    char sa_eye[4]; /* 4 byte identifier must be */
    int sa_len; /* length of the buffer to put data into*/
    char sa_flags; /* flags field must be x80 or x80, x80 means reset statistics*/
    #define SA_RESET 0x80
    char sa_fill[3]; /* spare bytes */
    int sa_reserve[4]; /* Reserved */
} stat_api_t;
```

---

z/OS V2R1.0 Distributed File Service zFS Administration
Statistics Locking Information

```c
int reserved1;
unsigned long stlk_untimed_sleeps; /* Number of untimed sleeps */
unsigned long stlk_timed_sleeps; /* Number of timed sleeps */
unsigned long stlk_wakeups; /* Number of wake ups */
unsigned long stlk_total_wait_for_locks; /* Total waits for locks */
int pad1;
double stlk_average_lock_wait_time; /*Average lock wait time */
int stlk_avg_lock_wait_time_whole; /*Average lock wait time in msecs */
int stlk_avg_lock_wait_time_decimal; /*Average lock wait time in msecs */
/*left of the decimal part */
/* in thousands */
/* for example, 3 means .003 and 300 means .3 */
unsigned long stlk_total_monitored_sleeps; /*Total monitored sleeps */
int pad2;
double stlk_average_monitored_sleep_time; /* Average monitored sleep time */
int stlk_avg_mon_sleep_time_whole; /*Average monitored sleep time in msecs */
int stlk_avg_mon_sleep_time_decimal; /*Average monitored sleep time in msecs */
/*left of the decimal part */
/* in thousands */
/* for example, 3 means .003 and 300 means .3 */
unsigned long stlk_total_contentions; /*Total lock contention of all kinds*/
char stlk_reserved_space[48]; /* reserved for future use */
int pad3;
#define MAX_LOCKS 15 /* Maximum number of locks in this release*/
#define MAX_SLEEPS 5 /* Maximum number of sleeps in this release*/
LOCK_LINE stlk_locks[MAX_LOCKS]; /* Storage for the lock data */
SLEEP_LINE stlk_sleeps[MAX_SLEEPS]; /* Storage for the top 5 most common sleep threads*/
} STAT_LOCKING;

/*********************************************************************/
/* The following structures are used to represent cgpq queries */
/* for locking stats */
/*********************************************************************/
struct parmstruct
{
    syscall_parmlist myparms;
    STAT_API myapi;
    STAT_LOCKING mystats;
    char systemname[9];
} myparmstruct;

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpaxs;
    int i;
    STAT_API *stapptr = &(myparmstruct.myapi);
    STAT_LOCKING *stlkptr = &(myparmstruct.mystats);
    char buf[33];
    myparmstruct.myparms.opcode = STATOP_LOCKING;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;
    memset(stapptr->sa_eye, SA_EYE, 4);
    stapptr->sa_ver=SA_VER_INITIAL;
    stapptr->sa_len=(int) sizeof(STAT_LOCKING);
    // Only specify a non-zero offset for the next field [parms[2]] if */
    // you are running z/OS 1.7 and above, and */
    // you want to query the locking statistics of a different system than this one */
    /* myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) + sizeof(STAT_LOCKING); */
    myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) + sizeof(STAT_LOCKING);
    memset(stapptr->sa_eye, SA_EYE, 4);
    stapptr->sa_ver=SA_VER_INITIAL;
    stapptr->sa_len=(int) sizeof(STAT_LOCKING);
    // This next field should only be set if parms[2] is non-zero */
    // strcpy(myparmstruct.systemname,"DCEIMGVQ"); */
    BXP1PCT("ZFS",
            ZSCALL_STATS, /* Perf statistics operation */
            sizeof(myparmstruct), /* Length of Argument */
            (char *)&myparmstruct, /* Pointer to Argument */
            &bpxrv, /* Pointer to Return_value */
            &bpxrc, /* Pointer to Return_code */
            &bpaxs); /* Pointer to Reason_code */
    if( bpxrv < 0 )
    {
        printf("Error querying locking stats, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpaxs);
        return bpxrc;
    }
}
```
Statistics Locking Information

else
 |
 printf("%55s\n","Locking Statistics");
 printf("\n");
 printf("\n");
 printf("Untimed sleeps: %10u Timed Sleeps: %10u Wakeups: %10u\n",
 myparmstruct.mystats.stlk_untimed_sleeps,
 myparmstruct.mystats.stlk_timed_sleeps,
 myparmstruct.mystats.stlk_wakeups);
 printf("\n");
 printf("Total waits for locks: %10u\n",
 myparmstruct.mystats.stlk_total_wait_for_locks);
 printf("Average lock wait time: %9u.%3.3u (msecs)\n",
 myparmstruct.mystats.stlk_avg_lock_wait_time_whole,
 myparmstruct.mystats.stlk_avg_lock_wait_time_decimal);
 printf("\n");
 printf("Top %u Most Highly Contended Locks\n",
 MAX_LOCKS);
 printf(" Thread Async Spin \n");
 printf(" Wait Disp. Resol. Pct. Description\n");
 printf("---------- ---------- ---------- ----- --------------
");
 for (i=0 ;i<MAX_LOCKS;i++ )
 |
 printf("%10u %10u %10u %3u.%1.1u%% %.80s\n",
 myparmstruct.mystats.stlk_locks[i].count,
 myparmstruct.mystats.stlk_locks[i].async,
 myparmstruct.mystats.stlk_locks[i].spins,
 myparmstruct.mystats.stlk_locks[i].percentage_whole,
 myparmstruct.mystats.stlk_locks[i].percentage_decimal,
 myparmstruct.mystats.stlk_locks[i].description);
 |
 printf("\n");
 printf("Total lock contention of all kinds: %10u\n",
 myparmstruct.mystats.stlk_total_contentions);
 printf("\n");
 printf("Top %u Most Common Thread Sleeps\n",
 MAX_SLEEPS);
 printf(" Thread Wait Pct. Description\n");
 printf("----------- ----- -----------
");
 for (i=0 ;i< MAX_SLEEPS;i++ )
 |
 printf("%10u %3u.%1.1u%% %.80s\n",
 myparmstruct.mystats.stlk_sleeps[i].sleepcount,
 myparmstruct.mystats.stlk_sleeps[i].percentage_whole,
 myparmstruct.mystats.stlk_sleeps[i].percentage_decimal,
 myparmstruct.mystats.stlk_sleeps[i].description);
 |
 printf("\n");
 if (0==ctime_r((time_t *) &stapptr->reset_time_infoposix_time_low, buf))
 |
 printf("Could not get timestamp.\n");
 |
 else
 |
 printf("Last Reset Time: %s",buf);
 |
 return 0;
 |

Statistics Log Cache Information

Purpose

This subcommand is a performance statistics operation that returns log cache counters, such as the number of requests, hits, and waits on the log buffer cache.
Statistics Log Cache Information

Format

```c
syscall_parmlist
opcode 247 STATOP_LOG_CACHE
parms[0] offset to STAT_API
parms[1] offset to output buffer
parms[2] offset to system name (optional)
parms[3] 0
parms[4] 0
parms[5] 0
parms[6] 0

STAT_API
sa_eye char[4] "STAP"
sa_len int length of buffer that follows STAT_API
sa_ver int 1
sa_flags char[1] 0x00
SA_RESET char[1] 0
sa_fill char[3] 0
sa_reserve int[4] 0
posix_time_high unsigned long high order 32 bits since epoch
posix_time_low unsigned long low order 32 bits since epoch
posix_useconds unsigned long microseconds

API_LOG_STATS
al_eye char[4] "ALOG"
al_size short size of output
al_version char version
al_reserved1 char reserved byte
al_reserved2 int reserved
al_buffersize int size of each buffer in K bytes
al_lookups_reserved int reserved
al_lookups int lookups/creates of item in log buffer cache
al_hits_reserved int reserved
al_hits int hits - number of items time item found in cache
al_writtenPagesReserved int reserved
al_writtenPages int number of log buffer pages written to disk
al_fullWaitsReserved int reserved
al_fullWaits int number of times new log buffer requires wait on prior
log pages
al_nbsWaitsReserved int reserved
al_nbsWaits int number of times new log buffer requires wait on new
block user I/O
al_reserved3 int[10] reserved
systemname char[9]

Return_value 0 if request is successful, -1 if it is not successful
Return_code
EINTR zFS is shutting down
EINVAL Invalid parameter list
ENOSRERR Internal error occurred
EZBSIG Information too big for buffer supplied
Reason_code
BKSMXXXX See z/OS Distributed File Service Messages and Codes
```

Usage notes
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required
None.

Related services
- Statistics Vnode Cache Information
- Statistics Metadata Cache Information

Restrictions
None.

Examples
```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int);

/* #include <stdlib.h> */
#include <stdio.h>
#define ZFSCALL_STATS 0x40000007
#define STATOP_LOG_CACHE 247 /* Performance API queries */
```
Statistics Log Cache Information

```c
#define u_long unsigned long
#define CONVERT_RATIO_TO_INTS(RATIO, INTEGER, DECIMAL) \
{ INTEGER = (int)(RATIO); \
  DECIMAL = (int)((RATIO - (double)INTEGER) * (double)1000.0); \
}

typedef struct syscall_parmlist_t 
{ int opcode; /* Operation code to perform */
  int parms[7]; /* Specific to type of operation. */
  /* provides access to the parms */
  /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct hyper 
{ unsigned long high; /* unsigned long reserved */
  unsigned long low; /* hyper */
} hyper;

typedef struct API_LOG_STATS_t 
{ char al_eye[4]; /* Eye catcher = ALOG */
  short al_size; /* Size of output structure */
  char al_version; /* Version of stats */
  #define LS_VER_INITIAL 1 /* First version of log stats */
  char al_reserved1; /* Reserved byte, 0 in version 1 */
  hyper al_buffers; /* Number of buffers used */
  int al_reserved2; /* Reserved for future use, 0 in version 1 */
  int al_buffsize; /* Size in kilobytes of one buffer */
  hyper al_lookups; /* Lookups/creates of item in log buffer cache */
  hyper al_hits; /* Hits, number of times item found in cache */
  hyper al_writtenPages; /* Number of log buffer pages written to disk */
  hyper al_fullWaits; /* Number of time new log buffer requires wait on prior log pages */
  int al_nbsWaits; /* Number of time new log buffer requires wait on new block user IO */
  int al_reserved[10]; /* Reserved for future use */
} API_LOG_STATS;

/* reset timestamp */
typedef struct reset_time 
{ u_long posix_time_high; /* high order 32 bits since epoc */
  u_long posix_time_low; /* low order 32 bits since epoch */
  u_long posix_usecs; /* microseconds */
  int pad1;
} RESET_TIME;

/* The following structure is the api query control block */
typedef struct stat_api_t 
{ #define SA_EYE "STAP" 
  struct syscall_parmlist_t myparms; /* 4 byte identifier must be */
  struct reset_time reset_time_info; /* lengths of the buffer to put data into */
  struct syscall_parmlist_t myapi; /* this buffer area follows this struct */
  char systemname[9]; /* the version number currently always 1*/
  int sa_len; /* flags field must be x80 or x80, x80 means reset statistics*/
  #define SA_RESET 0x80 
  char sa_flags; /* version number must be x80 or x80, x80 means reset statistics*/
  #define SA_VER_INITIAL 0x01 
  char sa_fill[3]; /* spare bytes */
  int sa_reserve[4]; /* Reserved */
  int sa_pad[5]; /* the version number currently always 1*/
} STAT_API;

struct parmstruct 
{ syscall_parmlist_t myparms; 
  STAT_API myapi; 
  API_LOG_STATS mystats; 
  char systemname[9]; 
} myparmstruct;

int main(int argc, char **argv) 
{ int bpxrv;
  int bpxrc;
  int bpxrs;
  int i;
  double temp_ratio;
  int whole, decimal;

  STAT_API *stapptr = &(myparmstruct.myparms);
  /* STAT_TRAN_CACHE *sttcptr = &(myparmstruct.mystats); */
  char buf[33];

  myparmstruct.myparms.opcode = STATOP_LOG_CACHE;
  myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
  myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
  myparmstruct.myparms.parms[2] = 0;

  Convert(300.0, whole, decimal);
```
Statistics Log Cache Information

Statistics Metadata Cache Information

Purpose

This subcommand call is a performance statistics operation that returns metadata cache counters. It is used to determine the number of requests, hits and discards from the directory cache.
Statistics Metadata Cache Information

Format

```c
syscall_parmlist
  opcode 248 STATOP_META_CACHE
  parms[0] offset to STAT_API
  parms[1] offset to output buffer
  parms[2] offset to system name (optional)
  parms[3] 0
  parms[4] 0
  parms[5] 0
  parms[6] 0

STAT_API
  sa_eye char[4] "STAP"
  sa_len int length of buffer that follows STAT_API
  sa_ver int 1
  sa_flags char[1] 0x00
  SA_RESET 0x80 Reset statistics
  sa_fill char[3] 0
  sa_reserve int[4] 0
  posix_time_high unsigned long high order 32 bits since epoch
  posix_time_low unsigned long low order 32 bits since epoch
  posix_useconds unsigned long microseconds
  pad1 int

API_META_STATS
  am_eye char[4] "AMET"
  am_size short size of output
  am_version char version
  am_reserved1 char reserved byte

PRIMARY_STATS
  buffers_reserved int reserved
  buffers int number of buffers in the cache
  bufsize int size of each buffer in K bytes
  amc_res1 int reserved
  requests reserved int reserved
  requests int requests to the cache
  hits_reserved int reserved
  hits int hits in the cache
  updates reserved int reserved
  updates int updates to buffers in the cache
  reserved int[10] reserved

BACK_STATS
  buffers hyper number of buffers in the cache
  bufsize int size of each buffer in K bytes
  amc_res1 int reserved
  requests reserved int reserved
  requests int requests to the cache
  hits_reserved int reserved
  hits int hits in the cache
  discards_reserved int reserved
  discards int discards of data from the cache
  reserved int[10] reserved

am_reserved3 int[10] reserved

systemname char[9]
```

Return_value 0 if request is successful, -1 if it is not successful

Return_code
- EINTR zFS is shutting down
- EINVAL Invalid parameter list
- EMWSERR Internal error occurred
- E2BIG Information too big for buffer supplied

Reason_code
- 0xefnnxxxx See z/OS Distributed File Service Messages and Codes

Usage notes
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required
- None.
Related services

Statistics Vnode Cache Information
Statistics Metadata Cache Information

Restrictions

None.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

/* #include <stdlib.h> */
#include <stdio.h>
#define ZFSCALL_STATS 0x40000007
#define STATOP_META_CACHE 248 /* Metadata cache (and back cache) stats */

#define u_long unsigned long
#define CONVERT_RATIO_TO_INTS(RATIO, INTEGER, DECIMAL) \
{ 
    INTEGER = (int)RATIO; 
    DECIMAL = (int)((RATIO - (double)INTEGER) * (double)1000.0); 
}

typedef struct syscall_parmlist_t 
{ 
    int opcode; /* Operation code to perform */ 
    int parms[7]; /* Specific to type of operation, */ 
    /* provides access to the parms */ 
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct hyper {
    unsigned long high; /* unsigned long reserved */
    unsigned long low; /* unsigned long reserved */
} hyper;

/***************************************************************************
/* META cache stats, including backing cache. */
***************************************************************************/

typedef struct PRIMARY_STATS_t 
{ 
    hyper buffers; /* Number of buffers in cache */
    int buffsize; /* Size of each buffer in k bytes */
    int amc_res1; /* Reserved for future use, zero in version 1 */
    int requests_reserved; /* Reserved */
    int requests; /* Requests to the cache */
    int hits_reserved; /* Reserved */
    int hits; /* Hits in the cache */
    int updates_reserved; /* Reserved */
    int updates; /* Updates to buffers in the cache */
    int reserved[10]; /* For future use */
} PRIMARY_STATS;

typedef struct BACK_STATS_t 
{ 
    hyper buffers; /* Number of buffers in cache */
    int buffsize; /* Size of each buffer in k bytes */
    int amc_res1; /* Reserved for future use, zero in version 1 */
    int requests_reserved; /* Reserved */
    int requests; /* Requests to the cache */
    int hits_reserved; /* Reserved */
    int hits; /* Hits in the cache */
    int discards_reserved; /* Reserved */
    int discards; /* Discards of data from backing cache */
    int reserved[10]; /* For future use */
} BACK_STATS;

typedef struct API_META_STATS_t 
{ 
    char am_eye[4]; /* Eye catcher = AMET */
    #define MS_EYE "AMET"
    short am_size; /* Size of output structure */
    char am_version; /* Version of stats */
    #define MS_VER_INITIAL 1 /* First version of log stats */
    char am_reserved1; /* Reserved byte, 0 in version 1 */
    PRIMARY_STATS am_primary; /* Primary space cache statistics */
    BACK_STATS am_back; /* Backing cache statistics */
    int am_reserved3[10]; /* Reserved */
} API_META_STATS;

/* reset timestamp */
typedef struct reset_time 
{ 
    u_long posix_time_high; /* High order 32 bits since epoch */
    u_long posix_time_low; /* Low order 32 bits since epoch */
    u_long posix_usecs; /* microseconds */
}
```
Statistics Metadata Cache Information

```c
int pad1;
} RESET_TIME;

/**************************************************************/
/* The following structure is the api query control block */
/* It is used for all api query commands */
/**************************************************************/
typedef struct stat_api_t
{
#define SA_EYE "STAP"
char sa_eye[4]; /* 4 byte identifier must be */
int sa_len; /* length of the buffer to put data into*/
/* this buffer area follows this struct*/
int sa_ver; /* the version number currently always 1*/
#define SA_VER_INITIAL 0x01
char sa_flags; /* flags field must be x00 or x80, x80 means reset statistics*/
#define SA_RESET 0x80
char sa_fill[3]; /* spare bytes */
int sa_reserve[4]; /* Reserved */
struct reset_time reset_time_info;
} STAT_API;

struct parmstruct
{
syscall_parmlist myparms;
STAT_API myapi;
API_META_STATS mystats;
char systemname[9];
} myparmstruct;

int main(int argc, char **argv)
{
int bpxrv;
int bpxrc;
int bpxrs;
int i;
double temp_ratio;
int whole,decimal;
STAT_API *stapptr = &(myparmstruct.myapi);
char buf[33];

myparmstruct.myparms.opcode = STATOP_META_CACHE;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
myparmstruct.myparms.parms[2] = 0;
// Only specify a non-zero offset for the next field (parms[2]) if */
// you are running z/OS 1.7 and above, and */
// you want to query the metadata cache statistics of a different system than this one */
// myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) + */
// sizeof(API_META_STATS);*/

myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) + */
// sizeof(API_META_STATS);*/

myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;
memset(stapptr,0,sizeof(STAT_API));
memcpy(stapptr->sa_eye,SA_EYE,4);
stapptr->sa_ver=SA_VER_INITIAL;
stapptr->sa_len=(int) sizeof(API_META_STATS);
// This next field should only be set if parms[2] is non-zero */
// strcpy(myparmstruct.systemname,"DCEIMGVQ"); */
BPX1PCT("ZFS *
ZFSSTATS, /* Perf statistics operation */
sizeof(myparmstruct), /* Length of Argument */
(char *) &myparmstruct, /* Pointer to Argument */
&bpxv, /* Pointer to Return_value */
&bpxc, /* Pointer to Return_code */
&bpxr); /* Pointer to Reason_code */
if( bpxrv < 0 )
{
printf("Error querying meta cache, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxc,bpxr);
return bpxrv;
}
else
{
/* Primary cache */
printf("%s\n" ,"Metadata Caching Statistics");
printf("Buffers (K bytes) Requests Hits Ratio Updates\n" );
printf("---------- --------------- \n" );
temp_ratio = (myparmstruct.mystats.am_primary.requests.low == 0) ? 0.0 :
((double)myparmstruct.mystats.am_primary.hits.low)/myparmstruct.mystats.am_primary.requests.low;
temp_ratio *= 100.0;
}
```

304 z/OS V2R1.0 Distributed File Service zFS Administration
Statistics Metadata Cache Information

```c
CONVERT_RATIO_TO_INTS(temp_ratio,whole, decimal);
decimal = decimal / 100; /* Just want tenths */
printf("%10u %9u %10u %10u %3u.%1.1u%% %10u
", myparmstruct.mystats.am_primary.buffers.low, myparmstruct.mystats.am_primary.buffers.low * myparmstruct.mystats.am_primary.buffsize, myparmstruct.mystats.am_primary.requests.low, myparmstruct.mystats.am_primary.requests.low, myparmstruct.mystats.am_primary.hits.low, whole, decimal, myparmstruct.mystats.am_primary.updates.low);

printf("\n");
#ifdef Backing cache
printf("%56s
","Metadata Backing Caching Statistics");
printf(" %");
printf("Buffers (K bytes) Requests Hits Ratio Discards\n");
printf("---------- --------- ---------- ---------- ------ ----------\n");
temp_ratio = (myparmstruct.mystats.am_back.requests.low == 0) ? 0.0 : ((double)myparmstruct.mystats.am_back.hits.low)/myparmstruct.mystats.am_back.requests.low;
temp_ratio *= 100.0;
CONVERT_RATIO_TO_INTS(temp_ratio,whole, decimal);
decimal = decimal / 100; /* Just want tenths */
printf("%10u %9u %10u %10u %3u.%1.1u%% %10u
", myparmstruct.mystats.am_back.buffers.low, myparmstruct.mystats.am_back.buffers.low * myparmstruct.mystats.am_back.buffsize, myparmstruct.mystats.am_back.requests.low, myparmstruct.mystats.am_back.requests.low, myparmstruct.mystats.am_back.hits.low, whole, decimal, myparmstruct.mystats.am_back.discards.low);
printf("\n");
if (0==ctime_r((time_t *) &stapptr->reset_time_info.posix_time_low, buf))
{
    printf("Could not get timestamp.\n");
}
else
{ /* Insert the microseconds into the displayable time value */
    strncpy(&buf[27],&buf[20],6);
    sprintf(&buf[20],"%06d",stapptr->reset_time_info.posix_usecs);
    buf[26]= ' '; 
    buf[19]= ' ',
    printf("Last Reset Time: %s",buf);
}
return 0;
}
```

Statistics Storage Information

**Purpose**

This subcommand call is a performance statistics operation that returns storage information.
Statistics Storage Information

Format

```
system_parmlist
  opcode          241    STATOP_STORAGE
  parm[0]         offset to STAT_API
  parm[1]         offset to STAT_STORAGE
  parm[2]         offset to system name (optional)
  parm[3]         0
  parm[4]         0
  parm[5]         0
  parm[6]         0

STAT_API
  sa_eye           char[4]    "STAP"
  sa_len          int         0
  sa_yer          int         1
  sa_flags        char        0x80 for reset; 0x00 otherwise
  sa_fill         char[3]     0
  sa_reserve      int[4]      0

API_STOR_STATS
  reserved1       int
  ss_total_bytes_allocated unsigned long /* Total bytes allocated*/
  ss_total_pieces_allocated unsigned long /* Total pieces allocated*/
  ss_total_allocation_requests unsigned long /* Total allocation requests*/
  ss_total_free_requests unsigned long /* Total free requests*/
  ss_number_of_comp_lines unsigned long /* Total number of components lines in buffer*/
  ss_reserved_space char[48]    /* reserved for future use */

COMP_LINE[n]
  ss_comp_bytes_allocated   int    /* The number of bytes allocated by this*/
                             /* component */
  ss_comp_pieces            int    /* The number of pieces allocated*/
  ss_comp_allocations       int    /* the number of storage allocations requests*/
                             /* done by this component */
  ss_comp_frees             int    /* the number of storage frees done by this*/
                             /* component */
  ss_comp_description       char[84]    /* the component description */

DETAIL_LINE[m]
  ss_detail_bytes_allocated int    /* number of bytes allocated*/
  ss_detail_pieces          int    /* number of pieces allocated*/
  ss_detail_allocations     int    /* number of allocation requests*/
  ss_detail_frees           int    /* number of free requests*/
  ss_detail_description     char[84]    /* description */

systemname     char[9]

Return value  0 if request is successful, -1 if it is not successful

Return code
  EINTR   ZFS is shutting down
  EINVAL  Invalid parameter list
  EMVSERR Internal error occurred
  E2BIG   Information too big for buffer supplied

Reason code
  0xEFxxnnnn  See z/OS Distributed File Service Messages and Codes
```

Usage notes

- You can specify a buffer that you think might be large enough or you can specify a buffer length of zero. If you get a return code E2BIG, the required size for the buffer is contained in the sa_len field.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

Statistics Locking Information
Statistics User Cache Information

Restrictions

None.
Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, char *, int, int, int);

#include <stdlib.h>
#include <stdio.h>

#define ZFSCALL_STATS 0x40000007
#define STATOP_LOCKING 240 /* Performance API queries */
#define u_long unsigned long

typedef struct syscall_parmlist_t
{
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    / parm[4]-parm[6] are currently unused*/
    } syscall_parmlist;

typedef struct Lock_line_t
{
    int count; /* Number of thread waits for this lock */
    int async; /* Asynchronous disposition*/
    int spins; /* Number of attempts to get lock that did not resolve immediately*/
    int pad1;
    double percentage; /* */
    int percentage_whole; /* percentage >= 1*/
    int percentage_decimal; /* percentage < 1 */
    /* in thousandths. */
    /* For example, 3 means .003 and 300 means .3 */
    char description[84]; /* Description of the lock */
    int pad2;
} LOCK_LINE;

typedef struct Sleep_line_t
{
    unsigned long sleepcount; /* Time spent sleeping */
    int pad1;
    double percentage; /* Percentage of time spent sleeping*/
    int percentage_whole; /* Percentage >=1 */
    int percentage_decimal; /* Percentage < 1 */
    /* in thousandths. */
    /* For example, 3 means .003 and 300 means .3 */
    char description[84]; /* Description of the thread*/
    int pad2;
} SLEEP_LINE;

/* reset timestamp */
typedef struct reset_time
{
    u_long posix_time_high; /* high order 32 bits since epoc */
    u_long posix_time_low; /* low order 32 bits since epoch */
    u_long posix_usecs; /* microseconds */
    int pad1;
} RESET_TIME;

/* The following structure is the api query control block */
/* It is used for all api query commands */
typedef struct stat_api_t
{
    #define SA_EYE "STAP" /* 4 byte identifier must be */
    char sa_eye[4];
    int sa_len; /* length of the buffer to put data into*/
    /* this buffer area follows this struct*/
    int sa_ver; /* the version number currently always 1*/
    #define SA_VER_INITIAL 0x01
    char sa_flags; /* flags field must be x00 or x80, x80 means reset statistics*/
    #define SA_RESET 0x80
    char sa_fill[3]; /* spare bytes */
    int sa_reserve[6]; /* Reserved */
    struct reset_time reset_time_info;
} STAT_API;

/* Statistics Storage Information */

Chapter 13. zFS application programming interfaces 307
unsigned long stlk_total_monitored_sleeps; /*Total monitored sleeps */
int pad2;
double stlk_avg_monitored_sleep_time; /* Average monitored sleep time */
int stlk_avg_mon_sleep_time_whole; /* Average monitored sleep time in msecs */
int stlk_avg_mon_sleep_time_decimal; /* Average monitored sleep time in msecs */
/* decimal portion */
/* in thousands */
/* for example, 3 means .003 and 300 means .3 */
unsigned long stlk_total_contentions; /*Total lock contention of all kinds*/
char stlk_reserved_space[48]; /* reserved for future use */
int pad3;
#define MAX_LOCKS 15 /* Maximum number of locks in this release*/
#define MAX_SLEEPS 5 /* Maximum number of sleeps in this release*/
LOCK_LINE stlk_locks[MAX_LOCKS]; /* Storage for the lock data */
SLEEP_LINE stlk_sleeps[MAX_SLEEPS]; /* Storage for the top 5 most common sleep threads*/
}
STAT_LOCKING;
/*********************************************************************/
/* The following structures are used to represent cfpop queries */
/* for locking stats */
/*********************************************************************/

struct parmstruct
{
    syscall_parmlist myparms;
    STAT_API myapi;
    STAT_LOCKING mystats;
    char systemname[9];
} myparmstruct;

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    int i;
    STAT_API *statptr = &(myparmstruct.myapi);
    STAT_LOCKING *stlkptr = &(myparmstruct.mystats);
    char buf[33];
    myparmstruct.myparms.opcode = STATOP_LOCKING;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;
    memset(statptr,0,sizeof(STAT_API));
    memcpy(statptr->sa_eye,SA_EYE,4);
    statptr->sa_ver=SA_VER_INITIAL;
    statptr->sa_len=(int) sizeof(STAT_LOCKING);
    /* This next field should only be set if parms[2] is non-zero */
    /* strcpy(myparmstruct.systemname,"DCEIMGVQ"); */

    BPKLPCT("ZFS",
    ZFSSTATS_STATS, /* Perf statistics operation */
    sizeof(myparmstruct), /* Length of Argument */
    (char *) &myparmstruct, /* Pointer to Argument */
    &bpxrv, /* Pointer to Return_value */
    &bpxrc, /* Pointer to Return_code */
    &bpxrs); /* Pointer to Reason_code */
    if( bpxrv < 0 )
    {
        printf("Error querying locking stats, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    }
    else
    {
        printf("%55s\n","Locking Statistics");
        printf("\n");
        printf("\n");
        printf("Untimed sleeps: %lu Timed Sleeps: %lu Wakeups: %lu\n", stlk_phys, stlk_timed, stlk_wakeups);
        printf("\n");
        printf("\n");
        printf("\n");
        printf("Total waits for locks: %lu\n", stlk_total_wait_for_locks);
        printf("\n");
        printf("\n");
        printf("\n");
    }
}
Statistics Transaction Cache Information

Purpose

This subcommand call is a performance statistics operation that returns transaction cache counters. It is used to determine the number of transactions in the transaction cache.
Statistics Transaction Cache Information

Format

```
system_parmlist
  opcode  250 STATOP_TRAN_CACHE
  parms[0] offset to STAT_API
  parms[1] offset to output buffer
  parms[2] offset to system name (optional)
  parms[3] 0
  parms[4] 0
  parms[5] 0
  parms[6] 0

STAT_API
  sa_eye     char[4] "STAP"
  sa_len     int   Length of buffer that follows STAT_API
  sa_year    int   1
  sa_flags   char[1] 0x00
  SA_RESET   int    0x80 Reset statistics
  sa_fill    char[3] 0
  sa_reserve int[4] 0
  posix_time_high unsigned long high order 32 bits since epoch
  posix_time_low unsigned long low order 32 bits since epoch
  posix_usec unsigned long microseconds

STAT_TRAN_CACHE
  sttr_started_high unsigned long transactions started high 32 bits
  sttr_started  unsigned long transactions started
  sttr_lookups_high unsigned long lookups on transaction high 32 bits
  sttr_lookups  unsigned long lookups on transaction
  sttr_ec_merges_high unsigned long equivalence class merges high 32 bits
  sttr_ec_merges unsigned long equivalence class merges
  sttr_alloc_trans_high unsigned long allocated transactions high 32 bits
  sttr_alloc_trans unsigned long allocated transactions
  sttr_trans_act_high unsigned long transactions active high 32 bits
  sttr_trans_act unsigned long transactions active
  sttr_trans_pend_high unsigned long transactions pending high 32 bits
  sttr_trans_pend unsigned long transactions pending
  sttr_trans_comp_high unsigned long transactions completed high 32 bits
  sttr_trans_comp unsigned long transactions completed
  sttr_trans_free_high unsigned long free transactions high 32 bits
  sttr_trans_free unsigned long free transactions
  reserved    char[60] reserved
  system_name char[9] system name

Return_value  0 if request is successful, -1 if it is not successful

Return_code
  EINTR   zFS is shutting down
  EINVAL  Invalid parameter list
  EMVSERR Internal error occurred
  E2BIG   Information too big for buffer supplied

Reason_code
  0xEFnnxxx See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

Statistics Vnode Cache Information
Statistics Metadata Cache Information

Restrictions

None.
Statistics Transaction Cache Information

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *
); /* #include <stdlib.h> */
#include <stdio.h>
define ZFSCALL_STATS 0x40000007
define STATOP_TRAN_CACHE 250 /* Performance API queries */
define u_long unsigned long
typedef struct syscall_parmlist_t
{
    int opcode; /* Operation code to perform */
    int parms[7]; /* Specific to type of operation, */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;
typedef struct stat_tran_cache_t
{
    unsigned long sttr_started_high;
    unsigned long sttr_started;
    unsigned long sttr_lookups_high;
    unsigned long sttr_lookups;
    unsigned long sttr_ec_merges_high;
    unsigned long sttr_ec_merges;
    unsigned long sttr_alloc_trans_high;
    unsigned long sttr_alloc_trans;
    unsigned long sttr_trans_act_high;
    unsigned long sttr_trans_act;
    unsigned long sttr_trans_pend_high;
    unsigned long sttr_trans_pend;
    unsigned long sttr_trans_comp_high;
    unsigned long sttr_trans_comp;
    unsigned long sttr_trans_free_high;
    unsigned long sttr_trans_free;
    char reserved[60];
} STAT_TRAN_CACHE;
/* reset timestamp */
typedef struct reset_time {
    u_long posix_time_high; /* high order 32 bits since epoc */
    u_long posix_time_low; /* low order 32 bits since epoch */
    u_long posix_usecs; /* microseconds */
    int pad1;
} RESET_TIME;
/**********************************************************/
/* The following structure is the api query control block */
/* It is used for all api query commands */
/**********************************************************/
typedef struct stat_api_t
{
    #define SA_EYE "STAP"
    char sa_eye[4]; /* 4 byte identifier must be */
    int sa_len; /* length of the buffer to put data into*/
    int sa_ver; /* this buffer area follows this struct*/
    /* the version number currently always 1*/
    #define SA_VER_INITIAL 0x01
    char sa_flags; /* flags field must be x00 or x80, x80 means */
    #define SA_RESET 0x80
    char sa_fill[3]; /* spare bytes */
    int sa_reserve[4]; /* Reserved */
    struct reset_time reset_time_info;
} STAT_API;

struct parmstruct
{
    syscall_parmlist myparms;
    STAT_API myapi;
    STAT_TRAN_CACHE mystats;
    char systemname[9];
} myparmstruct;

int main(int argc, char **argv)
{
```
Statistics Transaction Cache Information

```c
int bpxrv;
int bpxrc;
int bpxrs;
int i;

STAT_API *stapptr = &(myparmstruct.myapi);
STAT_TRAN_CACHE *sttcptr = &(myparmstruct.mystats);
char buf[33];

myparmstruct.myparms.opcode = STATOP_TRAN_CACHE;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
myparmstruct.myparms.parms[2] = 0;

/* Only specify a non-zero offset for the next field (parms[2]) if */
/* you are running z/OS 1.7 and above, and */
/* you want to query the tran cache statistics of a different system than this one */
/* myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) + */
/* sizeof(STAT_TRAN_CACHE); */

myparmstruct.myparms.parms[3] = 0;
myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;
memset(stapptr,0,sizeof(STAT_API));
memcpy(stapptr->sa_eye,SA_EYE,4);
stapptr->sa_ver=SA_VER_INITIAL;
stapptr->sa_len=(int) sizeof(STAT_TRAN_CACHE);

/* This next field should only be set if parms[2] is non-zero */
/* strcpy(myparmstruct.systemname,"DCEIMGVQ"); */

BPX1PCT("ZFS ",
  ZFSCALL_STATS, /* Perf statistics operation */
  sizeof(myparmstruct), /* Length of Argument */
  &(char *) &myparmstruct, /* Pointer to Argument */
  &bpxrv, /* Pointer to Return_value */
  &bpxrc, /* Pointer to Return_code */
  &bpxrs); /* Pointer to Reason_code */
if( bpxrv<0 )
{
  printf("Error querying tran cache, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
  return bpxrc;
}
else
{

  printf("Transaction Cache Statistics\n");
  printf("Trans started: %8u Lookups on Tran: %8u EC Merges: %8u\n",
    myparmstruct.mystats.sttr_started,
    myparmstruct.mystats.sttr_lookups,
    myparmstruct.mystats.sttr_ec_merges);
  printf("Allocated Trans: %8u (Act= %7u, Pend= %7u, Comp= %7u, Free= %7u)\n",
    myparmstruct.mystats.sttr_alloc_trans,
    myparmstruct.mystats.sttr_trans_act,
    myparmstruct.mystats.sttr_trans_pend,
    myparmstruct.mystats.sttr_trans_comp,
    myparmstruct.mystats.sttr_trans_free);

  if (0==ctime_r((time_t *) &stapptr->reset_time_info.posix_time_low, buf))
  {
    printf("Could not get timestamp.\n");
  }
  else
  {
    /* Insert the microseconds into the displayable time value */
    strncpy(buf[27],buf[20],5);
    sprintf(buf[20],"%06d",stapptr->reset_time_info.posix_usecs);
    buf[26]=',
    buf[19]=',
    printf("Last Reset Time: %s",buf);
  }
```

312  z/OS V2R1.0 Distributed File Service zFS Administration
Statistics Transaction Cache Information

Statistics User Cache Information

Purpose

This subcommand call is a performance statistics operation that returns user cache information.

Format

```c
syscall_parmlist
opcode 242 STATOP_USER_CACHE
parm[0] offset to STAT_API
parm[1] offset to STAT_USER_CACHE
parm[2] offset to system name (optional)
parm[3] 0
parm[4] 0
parm[5] 0
parm[6] 0

STAT_API
sa_eye char[4] "STAP"
sa_len int sizeof(STAT_USER_CACHE)
sa_ver int 1
sa_flags char 0x80 for reset; 0x00 otherwise
sa_fill char[3] 0
sa_reserve int[4] 0

STAT_USER_CACHE[2]
vm_schedules u_long
vm_setattrs u_long
vm_fsyncs u_long
vm_unmaps u_long
vm_reads u_long
vm_readsyncs u_long
vm_writes u_long
vm_getattrs u_long
vm_flushes u_long
vm_scheduled_deletes u_long
vm_unbound_read u_long
vm_unbound_reclaim u_long
vm_unbound_reclaim_wait u_long
vm_unbound_reclaim_steal u_long
vm_unbound_waits u_long
vm_error_writes u_long
vm_reclaim_writes u_long
vm_reclaim_waits u_long
vm_reclaim_steal u_long
vm_reclaim_wait u_long
vm_read_waits u_long
vm_write_waits u_long
vm_error_syncs u_long
vm_error_writes u_long
vm_error_waits u_long
vm_error_deletes u_long
vm_error_synchs u_long
vm_reclaim_deletes u_long
vm_reclaim_synchs u_long
vm_reclaim_deletes u_long
vm_waits_for_reclaim u_long

DS_ENTRY[32]
ds_name char[9]
pad1 char[3]
ds_alloc_segs int
ds_free_pages int
systemname char[9]
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.
Statistics User Cache Information

Privilege required

None.

Related services

Statistics Locking Information
Statistics Storage Information

Restrictions

None.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
#define ZFSCALL_STATS 0x40000007
#define STATOP_USER_CACHE 242 /* Performance API queries */
#define LOCAL 0
#define REMOTE 1
#define u_long unsigned long

typedef struct syscall_parmlist_t
{
    int opcode;    /* Operation code to perform */
    int parms[7];    /* Specific to type of operation. */
    /* provides access to the parms */
    /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct ds_entry
{
    char ds_name[9];
    char pad1[3];
    int ds_gloc_segs;
    int ds_free_pages;
    int ds_reserved[5];    /*reserved for future use*/
} DS_ENTRY;

typedef struct reset_time
{
    u_long posix_time_high;    /* high order 32 bits since epoch */
    u_long posix_time_low;     /* low order 32 bits since epoch */
    u_long posix_usecs;      /* microseconds */
    int pad1;
} RESET_TIME;

/*********************************************************************/
/* The following structure is the api query control block */
/* It is used for all api query commands */
/*********************************************************************/

typedef struct stat_api_t
{
    #define SA_EYE "STAP"
    char sa_eye[4];    /* 4 byte identifier must be */
    int sa_len;    /* length of the buffer to put data into*/
    /* this buffer area follows this struct*/
    int sa_ver;    /* the version number currently always 1*/
    #define SA_VER_INITIAL 0x01
    char sa_flags;    /* flags field must be x80 or x80, x80 means reset statistics*/
    #define SA_RESET 0x80
    char sa_fill[3];    /* spare bytes */
    int sa_reserve[4];    /* Reserved */
    struct reset_time reset_time_info;
} STAT_API;

/*********************************************************************/
/* The following structure is the user data cache statistics */
/*********************************************************************/

typedef struct vm_stats_t
{
    /**********************************************************************/
    /* First set of counters are for external requests to the VM system. */
    /**********************************************************************/
    u_long vm_schedules;
    u_long vm_setattrs;
    u_long vm_fsyncs;
    u_long vm_unmaps;
    u_long vm_reads;
    u_long vm_readasyncs;
```
Statistics User Cache Information

```c
u_long vm_writes;
u_long vm_getattrs;
u_long vm_flushes;
u_long vm_scheduled_deletes;

/**********************************************************************/
/* Next two are fault counters, they measure number of read or write */
/* requests requiring a fault to read in data, this synchronizes */
/* an operation to a DASD read, we want these counters as small as */
/* possible. (These are read I/O counters). */
/**********************************************************************/
u_long vm_reads_faulted;
u_long vm_writes_faulted;
u_long vm_read_ios;

/**********************************************************************/
/* Next counters are write counters. They measure number of times */
/* we scheduled and waited for write I/Os. */
/**********************************************************************/
u_long vm_scheduled_writes;
u_long vm_error_writes;
u_long vm_reclaim_writes; /* Wrote dirty data for reclaim */

/**********************************************************************/
/* Next counters are I/O wait counters. They count the number of */
/* times we had to wait for a write I/O and under what conditions. */
/**********************************************************************/
u_long vm_read_waits;
u_long vm_write_waits;
u_long vm_fsync_waits;
u_long vm_error_waits;
u_long vm_reclaim_waits; /* Waited for pending I/O for reclaim */

/**********************************************************************/
/* Final set are memory management counters. */
/**********************************************************************/
u_long vm_reclaim_steal; /* Number of times steal from others function invoked */
u_long vm_waits_for_reclaim; /* Waits for reclaim thread */

} VM_STATS;

typedef struct stat_user_cache_t
{
  VM_STATS stuc[2]; /*Various statistics for both LOCAL and REMOTE systems*/
  int stuc_dataspaces; /* Number of dataspaces in user data cache */
  int stuc_pages_per_ds; /* Pages per dataspaces */
  int stuc_seg_size_loc; /* Local Segment Size (in K) */
  int stuc_seg_size_rmt; /* Remote Segment Size (in K) */
  int stuc_page_size; /* Page Size (in K) */
  int stuc_cache_pages; /* Total number of pages */
  int stuc_total_free; /* Total number of free pages */
  int stuc_vmsegtable_cachesize; /* Number of segments */
  int stuc_reserved[5]; /* reserved */
  DS_ENTRY stuc_ds_entry[32]; /* Array of dataspaces entries */
} STAT_USER_CACHE;

struct parmstruct
{
  syscall_parmlist myparms;
  STAT_API myapi;
  STAT_USER_CACHE mystats;
  char systemname[9];
} myparmstruct;

int main(int argc, char **argv)
{
  int bpxrv;
  int bpxrc;
  int bpxrs;
  int i,j;
  double ratio1,ratio2,ratio3,ratio4;
  char buf[32];
  STAT_API *stapptr = &(myparmstruct.myapi);
  myparmstruct.myparms.opcode = STATOP_USER_CACHE;
  myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
  myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
  myparmstruct.myparms.parms[2] = 0;
  myparmstruct.myparms.parms[3] = 0;
  myparmstruct.myparms.parms[4] = 0;
  myparmstruct.myparms.parms[5] = 0;
  myparmstruct.myparms.parms[6] = 0;
  
  /* Only specify a non-zero offset for the next field (parms[2]) if */
  /* you are running z/OS 1.7 and above, and */
  /* you want to query the user cache statistics of a different system than this one */
  myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) + */
  /* sizeof(STAT_USER_CACHE); */
```
memset(stapptr,0,sizeof(STAT_API));
memcpy(stapptr->sa_eye,SA_EYE,4);
stapptr->sa_ver=SA_VER_INITIAL;
stapptr->sa_len=(int) sizeof(STAT_USER_CACHE);

/* This next field should only be set if parms[2] is non-zero */
/* strcpy(myparmstruct.systemname,"DCEIMGVQ"); */

BPX1PCT("ZFS",
    ZFSALL_STATS, /* Perf statistics operation */
    sizeof(myparmstruct), /* Length of Argument */
    (char *) &myparmstruct, /* Pointer to Argument */
    &bpxrv, /* Pointer to Return_value */
    &bpxrc, /* Pointer to Return_code */
    &bpxrs); /* Pointer to Reason_code */
if( bpxrv<0 )
{
    printf("Error querying user cache stats, BPXRV = %d BPXRC = %d BPXRS = %x
",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else
{
    printf(" User File (VM) Caching System Statistics
");
    printf(" ----------------------------------------
");
    for( i=0;i<=REMOTE ; i++ )
    {
        if( i == 0 )
        {
            printf(" Direct Statistics\n");
            printf(" -----------------
");
            printf("\n");
        }
        else
        {
            printf(" Client Statistics\n");
            printf(" -----------------
");
            printf("\n");
        }
        printf("External Requests:\n");
        printf("File System Reads:\n");
        printf("File System Writes:\n");
        printf("\n");
        printf("Reads %9u Fsyncs %9u Schedules %9u\n", myparmstruct.mystats.stuc[i].vm_reads, myparmstruct.mystats.stuc[i].vm_fsyncs, myparmstruct.mystats.stuc[i].vm_schedules);
        ratio1 = ratio1 + myparmstruct.mystats.stuc[i].vm_read_ios;
        if( myparmstruct.mystats.stuc[i].vm_read_ios > 0 )
            ratio1 = 100 * (double)myparmstruct.mystats.stuc[i].vm_read_ios / (double)myparmstruct.mystats.stuc[i].vm_reads;
        if( myparmstruct.mystats.stuc[i].vm_read_faults > 0 )
        {
            ratio0 = 100 * (double)myparmstruct.mystats.stuc[i].vm_read_faults / (double)myparmstruct.mystats.stuc[i].vm_reads;
        }
        if( myparmstruct.mystats.stuc[i].vm_read_waits > 0 )
        {
            ratio3 = 100 * (double)myparmstruct.mystats.stuc[i].vm_read_waits / (double)myparmstruct.mystats.stuc[i].vm_reads;
        }
        printf("\n");
        printf("File System Reads:\n");
        printf("\n");
        printf("Scheduled Writes %10u Sync Waits %10u\n", myparmstruct.mystats.stuc[i].vm_scheduled_writes, myparmstruct.mystats.stuc[i].vm_fsync_waits);
        printf("\n");
        printf("Error Writes %10u Error Waits %10u\n", myparmstruct.mystats.stuc[i].vm_error_writes, myparmstruct.mystats.stuc[i].vm_error_waits);
        printf("\n");
        printf("\n");
    }
}
Statistics Vnode Cache Information

Purpose

This subcommand call is a performance statistics operation that returns vnode cache counters. It is used to determine the number of requests, hits and discards from the vnode cache.
Statistics Vnode Cache Information

Format

syscall_parmlist
opcode 251 STATOP_VNODE_CACHE
parms[0] offset to STAT_API
parms[1] offset to output buffer
parms[2] offset to system name (optional)
parms[3] 0
parms[4] 0
parms[5] 0
parms[6] 0

STAT_API
sa_eye char[4] "STAP"
sa_len int length of buffer that follows STAT_API
sa_ver int 1
SA_RESET 0x80 Reset statistics
sa_fill char[3] 0
sa_reserve int[4] 0
posix_time_high unsigned long high order 32 bits since epoch
posix_time_low unsigned long low order 32 bits since epoch
posix_useconds unsigned long microseconds

STAT_VNODE_CACHE
VNODE_STATS_API_STRUCT
reserved hyper reserved
Vnodes hyper number of vnodes
Requests hyper number of requests
Hits hyper number of hits
RatioWhole hyper ratio of hits to requests (whole number part)
RatioDecimal hyper ratio of hits to requests (decimal part)
decimal part is in thousanths
3 means .003 and 300 means .3
Allocates hyper allocates
Deletes hyper deletes
VnodeStructSize hyper base vnode structure size
ExtendedVnodes hyper number of extended vnodes
extensionSize hyper size of vnode extension
USShieldVnodes hyper number of held vnodes
USShieldVnodesHi hyper hi water mark of held vnodes
OpenVnodes hyper number of open vnodes
OpenVnodesHi hyper hi water mark of open vnodes
OpenVnodesReuse hyper number of vnodes that can be reused
reserved1 long[3] reserved
pad1 int padding
reserved2 hyper[10] reserved

EFS_STATS_API_STRUCT
reserved hyper reserved
grand_total_vnodes hyper total count of vnode ops
total_ops hyper number of vnode op counts
reserved1 long[3] reserved
pad1 int reserved
reserved2 hyper[10] reserved

ZFS_VNODED_OP_COUNTS
opname char[26] vnode operation name
pad1 char[2] reserved
opcount hyper count of vnode op requests
reserved hyper[2] reserved
reserved hyper[10] reserved
systemname char[9]

Return_value 0 if request is successful, -1 if it is not successful

Return_code
EINTR zFS is shutting down
EINVAL Invalid parameter list
EMVSERR Internal error occurred
E2BIG Information too big for buffer supplied

Reason_code
0xEFnnxxxx See z/OS Distributed File Service Messages and Codes

Usage notes
- Reserved fields and undefined flags must be set to binary zeros.
Statistics Vnode Cache Information

Privilege required

None.

Related services
Statistics Vnode Cache Information
Statistics Metadata Cache Information

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

/* #include <stdlib.h> */
#include <stdio.h>
define ZFS_CALL_STATS 0x40000007
#define STATOP_VNODE_CACHE 251 /* vnode cache stats */
define u_long unsigned long

define CONVERT_RATIO_TO_INTS(RATIO, INTEGER, DECIMAL) \ 
{ \ 
INTEGER = (int)RATIO; \ 
DECIMAL = (int)((RATIO - (double)INTEGER) * (double)1000.0); \ 
}

typedef struct syscall_parmlist_t 
{ \ 
int opcode; /* Operation code to perform */ \ 
int parms[7]; /* Specific to type of operation. */ \ 
/* provides access to the parms */ \ 
/* parms[4]-parms[6] are currently unused */ \ 
syscall_parmlist; 
} syscall_parmlist;

typedef struct hyper 
unsigned long high; /* unsigned long reserved */ \ 
unsigned long low; 
} hyper;

/* reset timestamp */
typedef struct reset_time 
{ \ 
u_long posix_time_high; /* high order 32 bits since epoc */ \ 
u_long posix_time_low; /* low order 32 bits since epoch */ \ 
u_long posix_usecs; /* microseconds */ \ 
int pad1; 
} RESET_TIME;

/* API STATOP_VNODE_CACHE storage structures */
typedef struct VNM_STATS_API_STRUCT_T 
{ \ 
hyper reserved; \ 
hyper Vnodes; \ 
hyper Requests; \ 
hyper Hits; \ 
hyper RatioWhole; /* decimal part is in thousands */ \ 
hyper RatioDecimal; /* 3 means .003 and 300 means .3 */ \ 
hyper Allocates; \ 
hyper Deletes; \ 
hyper VnodeStructSize; \ 
hyper ExtendedVnodes; \ 
hyper extensionSize; /* (minimum) in bytes */ \ 
hyper USHHeldVnodes; \ 
hyper USHHeldVnodesHi; \ 
hyper OpenVnodes; \ 
hyper OpenVnodesHi; \ 
hyper OpenVnodesReuse; \ 
long reserved[3]; \ 
int pad1; \ 
hyper reserved[16]; 
} VNM_STATS_API_STRUCT;

typedef struct ZFSVNODEOPCOUNTS_T 
{ \ 
char opname[26]; /* Operation being counted */ \ 
char pad1[2]; \ 
hyper opcount; /* Number of operations performed */ \ 
hyper reserved[2]; /* reserved for future use */ \ 
} ZFSVNODEOPCOUNTS;
```
typedef struct EFS_STATS_API_STRUCT_T
{
    hyper reserved;
    hyper grand_total_vnodes;
    hyper total_ops;
    long reserved1[3];
    int pad1;
    hyper reserved2[10];
    ZFSVNODEOPCROUNTS zFSOpCounts[50];
} EFS_STATS_API_STRUCT;

typedef struct stat_vnode_cache_t
{
    VM/stats_API_STRUCT vm.stats_info;
    EFS_STATS_API_STRUCT efs.stats_info;
    hyper reserved[10];
} STAT_VNODE_CACHE;

/**************************************************************************/
/* The following structure is the api query control block */
/* It is used for all api query commands */
/**************************************************************************/

typedef struct stat_api_t
{
    #define SA_EYE "STAP"
    char sa.eye[4]; /* 4 byte identifier must be */
    int sa.len; /* length of the buffer to put data into*/
    int sa.version; /* this buffer area follows this struct*/
    char sa.version; /* the version number currently always 1*/
    #define SA_VER_INITIAL 0x01
    char sa.flags; /* flags field must be x80 or x80, x80 means reset statistics*/
    #define SA_RESET 0x80
    char sa.fill[3]; /* spare bytes */
    int sa.reserve[4]; /* Reserved */
    struct reset_time reset_time_info;
} STAT_API;

struct parmstruct
{
    syscall_parmlist myparms;
    STAT_API myapi;
    STAT_VNODE_CACHE mystats;
    char systemname[9];
} myparmstruct;

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    int i;
    double temp_ratio;
    int whole, decimal;

    STAT_API *stapptr = &(myparmstruct.myapi);
    char buf[33];

    myparmstruct.myparms.opcode = STATOP_VNODE_CACHE;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
    myparmstruct.myparms.parms[2] = 0;

    /* Only specify a non-zero offset for the next field (parms[2]) if */
    /* you are running z/OS 1.7 and above, and */
    /* you want to query the vnode cache statistics of a different system than this one */
    /* myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) + */
    /* sizeof(STAT_VNODE_CACHE); */
    /* */
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;
    memset(stapptr,0,sizeof(STAT_API));
    memcpy(stapptr->sa.eye,SA_EYE,4);
    stapptr->sa.version=SA_VER_INITIAL;
    stapptr->sa.len=(int) sizeof(STAT_VNODE_CACHE);

    /* This next field should only be set if parms[2] is non-zero */
    /* strcpy(myparmstruct.systemname,"OEIMMVQ"); */
    /* */
    BPK1PCT("ZFS",
        ZFSZSCALL_STATS, /* Perf statistics operation */
        sizeof(myparmstruct), /* Length of Argument */
        (char *)&myparmstruct, /* Pointer to Argument */
        &bpxrv, /* Pointer to Return_value */
        &bpxrc, /* Pointer to Return_code */
        &bpxrs); /* Pointer to Reason_code */
    if( bpxrv < 0 )
Statistics Vnode Cache Information


def _ (Error querying vnode cache, BPXRV = %d BPXRC = %d BPXRS = %x
n",bpNrv,bpNrc,bpNrs); return bpNrc;
}
else{
i=0;
printf("%50s\n","zFS Vnode Op Counts
n");
printf("\n");
printf("Vnode Op Count Vnode Op Count
n------------------------ ---------- ------------------------ ---------- 
\\nwhile (i<=myparmstruct.mystats.efs_stats_info.total_ops.low)
|
| printf("%-25s %10u
n",myparmstruct.mystats.efs_stats_info.zFSOpCounts[i].opname,
| myparmstruct.mystats.efs_stats_info.zFSOpCounts[i].opcount.low);
| if (i<=myparmstruct.mystats.efs_stats_info.total_ops.low)
|  printf("%-25s %10u\n",myparmstruct.mystats.efs_stats_info.zFSOpCounts[i].opname,
| myparmstruct.mystats.efs_stats_info.zFSOpCounts[i].opcount.low);
| }
| printf("nTotal zFS Vnode Ops %10u\n",myparmstruct.mystats.efs_stats_info.grand_total_vnodes.low);
| printf("%52s\n","zFS Vnode Cache Statistics
n");
| printf("\n");
| printf(" Vnodes Requests Hits Ratio Allocates Deletes\n
n");
| printf(" -------------------------- ------------------------ ----

\n| printf("%-25s %10u %10u %10u %3u.%1.1u%% %10u %10u
n",myparmstruct.mystats.efs_stats_info.Vnodes.low,
| myparmstruct.mystats.efs_stats_info.Requests.low,
| myparmstruct.mystats.efs_stats_info.Hits.low,
| myparmstruct.mystats.efs_stats_info.RatioWhole.low,
| myparmstruct.mystats.efs_stats_info.RatioDecimal.low,
| myparmstruct.mystats.efs_stats_info.Allocates.low,
| myparmstruct.mystats.efs_stats_info.Deletes.low);
| printf("\n");
| printf("zFS Vnode structure size: %u bytes\n",myparmstruct.mystats.efs_stats_info.VnodeStructSize.low);
| printf("zFS extended vnodes: %u, extension size %u bytes (minimum)\n",myparmstruct.mystats.efs_stats_info.ExtendedVnodes.low,
| myparmstruct.mystats.efs_stats_info.ExtendedVnodesSize.low);
| printf("Held zFS vnodes: %10u (high %10u) inOpen zFS vnodes: %10u (high %10u) Reusable: %u\n",myparmstruct.mystats.efs_stats_info.USSHeldVnodes.low,
| myparmstruct.mystats.efs_stats_info.USSHeldVnodesHi.low,
| myparmstruct.mystats.efs_stats_info.OpenVnodes.low,
| myparmstruct.mystats.efs_stats_info.OpenVnodesHi.low,
| myparmstruct.mystats.efs_stats_info.OpenVnodesReuse.low);
| printf("\n");
| if (0==ctime_r((time_t *) &stapptr->reset_time_info.posix_time_low, buf))
|  printf("Could not get timestamp.\n");
| } else
|  /* Insert the microseconds into the displayable time value */
|  strncpy(&buf[27],&(buf[20]),6);
|  sprintf(&(buf[20]),"%06d",stapptr->reset_time_info.posix_usecs);
|  buf[26]=' ';
|  buf[19]='.';
|  printf("Last Reset Time: %s\n",buf);
|}
return 0;

Unquiesce Aggregate

Purpose

This subcommand call is an aggregate operation that unquiesces a zFS compatibility mode aggregate on a system. This subcommand call allows activity on the aggregate and its file system to resume.
Unquiesce Aggregate

Format

```c
typedef struct syscall_parmlist_t {
    int opcode;  /* Operation code to perform */
    int parms[7];  /* Specific to type of operation, provides access to the parms */
    /* parms[4]-parms[6] are currently unused */
} syscall_parmlist;
```

```c
#define ZFS_MAX_AGGRNAME 44
```

```
typedef struct aggr_id_t {
    char aid_eye[4];  /* Eye catcher */
    char aid_len;  /* Length of this structure */
    char aid_ver;  /* Version */
    char aid_name[45];  /* OMVS.PRV.AGGR001.LDS0001 */
    char aid_reserved[33] 0
} aggr_id_t;
```

Usage notes

- The unquiesce call must supply the quiesce handle that was returned by the quiesce call. The aggregate is typically quiesced before backing up the aggregate. After the backup is complete, the aggregate can be unquiesced.

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

- Quiesce Aggregate

Restrictions

None.

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
```

```c
#include <stdio.h>
#include <stdlib.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_UNQUIESCE_PARMDATA 133
```

```c
typedef struct syscall_parmlist_t {
    int opcode;  /* Operation code to perform */
    int parms[7];  /* Specific to type of operation, provides access to the parms */
    /* parms[4]-parms[6] are currently unused */
} syscall_parmlist;
```

```c
#define ZFS_MAX_AGGRNAME 44
```

```c
typedef struct aggr_id_t {
    char aid_eye[4];  /* Eye catcher */
    char aid_len;  /* Length of this structure */
    char aid_ver;  /* Version */
    char aid_name[45];  /* OMVS.PRV.AGGR001.LDS0001 */
    char aid_reserved[33] 0
} aggr_id_t;
```

Return_value 0 if request is successful, -1 if it is not successful

Return_code

EINTR ZFS is shutting down
EMVSERR Internal error using an osi service
ENODATA Aggregate is not attached
EPERM Permission denied to perform request

Reason_code

0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
Unquiesce Aggregate

```c
char aid_name[ZFS_MAX_AGGNAME+1]; /* Name, null terminated */
char aid_reserved[33]; /* Reserved for the future */
}

struct parmstruct
{
    syscall_parmlist myparms;
    AGGR_ID aggr_id;
};

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";
    long save_quiesce_handle;
    struct parmstruct myparmstruct;
    if (argc != 2)
    {
        printf("This unquiesce program requires a quiesce handle from the quiesce program as a parameter\n");
        return 1;
    }
    save_quiesce_handle = atoi(argv[1]);
    myparmstruct.myparms.opcode = AGOP_UNQUIESCE_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = save_quiesce_handle;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;
    memset(&myparmstruct.aggr_id,0,sizeof(AGGR_ID)); /* Ensure reserved fields are 0 */
    memcpy(&myparmstruct.aggr_id.aid_eye,AID_EYE,4);
    myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
    myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
    strcpy(myparmstruct.aggr_id.aid_name,aggrname);
    /* Aggregate operation */
    sizeof(myparmstruct), /* Length of Argument */
    (char *) &myparmstruct, /* Pointer to Argument */
    &bpxrv, /* Pointer to Return_value */
    &bpxrc, /* Pointer to Return_code */
    &bpxrs); /* Pointer to Reason_code */
    if (bpxrv < 0)
    {
        printf("Error unquiescing aggregate %s\n", aggrname);
        printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    }
    else /* Return from unquiesce was successful */
    {
        printf("Aggregate %s unquiesced successfully\n",aggrname);
    }
    return 0;
}
```
Unquiesce Aggregate
Part 3. Appendixes
Appendix A. Running the zFS pfsctl APIs in 64-bit mode

The pfsctl (BPX1PCT) application programming interface can be invoked in a 64-bit environment. To do this, you must take the following steps:

1. Replace the BPX1PCT with BPX4PCT
2. Replace the `#pragma linkage(BPX1PCT, OS)` statement with `#pragma linkage(BPX4PCT, OS64_NOSTACK)`
3. Change all the "long" declares to "int"
4. Ensure there are appropriate includes for function calls
5. Ensure all functions requiring 64-bit parameters are passing 64-bit numbers (for example, `ctime_r`).

The remaining code is, or can remain, unchanged. "Statistics iocounts information (64-bit mode)" shows example code that has been updated to be invoked in a 64-bit environment.

Statistics iocounts information (64-bit mode)

Examples

```c
#pragma linkage(BPX4PCT, OS64_NOSTACK)
extern void BPX4PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>
#include <time.h>
define ZFSCALL_STATS 0x40000007
#define STATOP_IOCOUTS 243 /* Performance API queries */
define TOTAL_TYPES 3
define TOTAL_CIRC 18
define u_int unsigned int
typedef struct syscall_parmlist_t
{  int opcode;   /* Operation code to perform */  int parms[7]; /* Specific to type of operation, */  /* provides access to the parms */  /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;
typedef struct reset_time {
  u_int posix_time_high; /* high order 32 bits since epoc */  u_int posix_time_low; /* low order 32 bits since epoch */  u_int posix_usec; /* microseconds */  int pad1;
} RESET_TIME;
/* The following structure is the api query control block */
/* It is used for all api query commands */
typedef struct stat_api_t
{  #define SA_EYE "STAP"  char sa_eye[4]; /* 4 byte identifier must be */  int sa_len; /* length of the buffer to put data into*/  int sa_ver; /* this buffer area follows this struct*/  /* the version number currently always 1*/  char sa_flags; /* flags field must be x80 or x80, x80 means reset statistics*/  #define SA_RESET 0x80  char sa_fill[3]; /* spare bytes */  int sa_reserve[4]; /* Reserved */
```

© Copyright IBM Corp. 2001, 2013
Statistics iocounts information (64-bit) mode

```c
struct reset_time reset_time_info;
} STAT_API;

typedef struct API_IO_BY_TYPE_t {
    unsigned int number_of_lines;
    unsigned int count;
    unsigned int waits;
    unsigned int cancels; /* Successful cancels of IO */
    unsigned int merges; /* Successful merges of IO */
    char reserved1[6];
    char description[51];
    char pad1[3];
} API_IO_BY_TYPE;

typedef struct API_IO_BY_CIRC_t {
    unsigned int number_of_lines;
    unsigned int count;
    unsigned int waits;
    unsigned int cancels;
    unsigned int merges;
    char reserved1[6];
    char description[51];
    char pad1[3];
} API_IO_BY_CIRC;

/*********************************************************************/
/* The following structures are used to represent cfgop queries */
/* for iocounts */
/*********************************************************************/

struct parmstruct {
    syscall_parmlist myparms;
    STAT_API myapi;
    API_IO_BY_TYPE mystatsbytype[TOTAL_TYPES];
    API_IO_BY_CIRC mystatsbycirc[TOTAL_CIRC];
} myparmstruct;

int main(int argc, char **argv) {
    int bpxrv;
    int bpxrc;
    int bpxrs;
    int i;
    STAT_API *stapptr = &(myparmstruct.myapi);
    API_IO_BY_TYPE *stiotptr = &(myparmstruct.mystatsbytype[0]);
    API_IO_BY_CIRC *stiocptr = &(myparmstruct.mystatsbycirc[0]);

    char buf[33];

    myparmstruct.myparms.opcode = STATOP_IOCOUNTS;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    memset(stapptr,0,sizeof(STAT_API));
    memcpy(stapptr->sa_eye,SA_EYE,4);
    stapptr->sa_ver=SA_VER_INITIAL;
    stapptr->sa_len=(int) (TOTAL_TYPES * sizeof(API_IO_BY_TYPE)) + (TOTAL_CIRC * sizeof(API_IO_BY_CIRC));

    BPX4PCT("ZFS",
        ZFSCALL_STATS, /* Perf statistics operation */
        sizeof(myparmstruct), /* Length of Argument */
        (char *) &myparmstruct, /* Pointer to Argument */
        &bpxrv, /* Pointer to Return_value */
        &bpxrc, /* Pointer to Return_code */
        &bpxrs); /* Pointer to Reason_code */

    if( bpxrv<0 ) {
        printf("Error querying iocounts, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    }
}
```
else
{
    if( stiotptr->number_of_lines != TOTAL_TYPES )
    {
        printf("Unexpected number of IO Types, %d instead of TOTAL_TYPES\n", stiotptr->number_of_lines);
        return 1;
    }
    if( stiocptr->number_of_lines != TOTAL_CIRC )
    {
        printf("Unexpected number of IO Circumstances, %d instead of TOTAL_CIRC\n", stiocptr->number_of_lines);
        return 2;
    }
    printf(" I/O Summary By Type\n");
    printf("---------- ------------ ------------");
    printf("\n");
    printf("Count Waits Cancels Merges Type\n");
    printf("---------- ------------ ------------\n");
    for( i=0; i<TOTAL_TYPES; i++ )
    {
        printf("%10u %10u %10u %10u %s\n", stiotptr->count, stiotptr->waits, stiotptr->cancels, stiotptr->merges, stiotptr->description);
        stiotptr = stiotptr + 1;
    }
    printf("\n");
    printf(" I/O Summary By Circumstance\n");
    printf("---------- ------------ ------------\n");
    printf("\n");
    printf("Count Waits Cancels Merges Circumstance\n");
    printf("---------- ------------ ------------\n");
    for( i=0; i<TOTAL_CIRC; i++ )
    {
        printf("%10u %10u %10u %10u %s\n", stiocptr->count, stiocptr->waits, stiocptr->cancels, stiocptr->merges, stiocptr->description);
        stiocptr = stiocptr + 1;
    }
    if (0==ctime_r((time_t *) &stapptr->reset_time_info, buf))
    {
        printf("Could not get timestamp.\n");
    }
    else
    {
        /* Insert the microseconds into the displayable time value */
        strncpy(&buf[27],&buf[20],6);
        sprintf(&buf[20],"%06d\n",stapptr->reset_time_info.posix_usecs);
        buf[26]= ' ';
        buf[19]='.';
        printf("Last Reset Time: %s\n",buf);
    }
    return 0;
}
Statistics iocounts information (64-bit) mode
Appendix B. Accessibility

Accessible publications for this product are offered through the z/OS Information Center, which is available at www.ibm.com/systems/z/os/zos/bkserv/.

If you experience difficulty with the accessibility of any z/OS information, please send a detailed message to mhvrcfs@us.ibm.com or to the following mailing address:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size.

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to z/OS TSO/E Primer, z/OS TSO/E User’s Guide, and z/OS ISPF User’s Guide Vol I for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the z/OS Information Center using 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 can be considered as 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 your 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, you
know that 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 can be used 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 giving 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, this indicates a reference that is
defined elsewhere. The string following the % symbol is the name of a syntax
fragment rather than a literal. For example, the line 2.1 %OP1 means that you
should refer to separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:
• ? means an optional syntax element. A dotted decimal number followed by the ?
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 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.
• ! means a default syntax element. A dotted decimal number followed by the !
symbol and a syntax element indicates 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 same dotted decimal number can
specify a ! 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 this example, if you include the FILE keyword but do not specify an option,
default option KEEP will be applied. A default option also applies to the next
higher dotted decimal number. In this example, if the FILE keyword is omitted,
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 only applies 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.

- * means a syntax element that can be repeated 0 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, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

**Note:**
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 could write HOST STATE, but you could not write HOST HOST.
3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

- + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this 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 only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back 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 disclaimers of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel  
IBM Corporation  
2455 South Road  
Poughkeepsie, NY 12601-5400  
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted
for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

**Minimum supported hardware**

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS](http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM representative.

**Programming Interface Information**

This publication, [*z/OS Distributed File Service zFS Administration*](http://www.ibm.com/software/support/systemsz/lifecycle/) primarily documents information that is NOT intended to be used as Programming Interfaces of the Distributed File Service.

*z/OS Distributed File Service zFS Administration* also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of the Distributed File Service. This information is identified where it occurs by an introductory statement to a chapter or section or by the following marking:

[- - NOT Programming Interface information ---]
[- - End of NOT Programming Interface information ---]

**Trademarks**

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)

UNIX is a registered trademark of The Open Group in the United States and other countries.
Glossary

This glossary includes terms and definitions for Distributed File Service z/OS File System. The following cross-references are used in this glossary:

1. See refers the reader from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
2. See also refers the reader to a related or contrasting term.

To view glossaries for other IBM products, go to www.ibm.com/software/globalization/terminology.

aggregate
A structured collection of data objects that form a data type.

attach
In z/OS, to create a task that can execute concurrently with the attaching code.

audit identifier
In zFS, a 16-byte value associated with each z/OS UNIX file or directory that provides identity in an SMF audit record or in certain authorization failure messages.

bitmap
In zFS, a file listing the blocks that are free on disk. The file size is dependent on the size of the aggregate.

catch-up mount
A local mount that z/OS UNIX automatically issues to every other system’s physical file system that is running sysplex-aware for that mode (read-write or read-only) when a sysplex-aware file system mount is successful on a system in a shared file system environment.

compatibility mode aggregate
A Virtual Storage Access Method linear data set (VSAM LDS) that contains a single read-write zFS file system.

DFS
See Distributed File Service.

Distributed File Service (DFS)
A base element of z/OS that allows users to access and share data in a distributed environment across a wide range of IBM and non-IBM platforms.

EAV
See extended address volume.

extended address volume (EAV)
DASD storage that can contain more than 65,521 cylinders per volume.

file handle
A number that is used by the client and server sides of the Network File System (NFS) or the Server Message Block (SMB) to specify a particular file or prefix.

file system owner
In z/OS, the system that coordinates sysplex activity for a particular file system.

function shipping
The process of requesting function from the owning file system and returning the response to the requester through XCF communications.

global resource serialization
A component of z/OS that serializes the use of system resources and converts hardware reserves on direct access storage device (DASD) volumes to data set enqueues.

global resource serialization complex
A group of systems that use global resource serialization to serialize access to shared resources such as data sets on shared direct access storage device (DASD) volumes.

hang
To become unresponsive to user commands and to stop or appear to stop processing.

i-node
The internal structure that describes the individual files in the UNIX file system. An i-node contains the node, type, owner, and location of a file.

local mount
A mount that is known to the physical file system.

metadata
Data that describes the characteristics of data; descriptive data.

non-sysplex aware
A mounted file system that has file
requests handled by remotely function shipping requests through z/OS UNIX

**root file system**
The basic file system onto which all other file systems can be mounted. The root file system contains the operating system files that run the rest of the system.

**salvager**
In zFS, a program that examines a zFS aggregate to determine if there are any inconsistencies in the structure of the aggregate.

**Server Message Block (SMB)**
A protocol that manages requests and responses in a client/server environment so that clients on a network can share files, directories, and devices.

**SMB** See [Server Message Block](#)

**sysplex**
A set of z/OS systems that communicate with each other through certain multisystem hardware components and software services.

**sysplex-aware**
A mounted file system that has file requests handled locally instead of function shipping requests through z/OS UNIX.

**version file system**
See [root file system](#)

**zFS** See [z/OS file system](#)

**zFS aggregate**
A Virtual Storage Access Method Linear Data Set (VSAM LDS) that contains a zFS file system.

**z/OS File System (zFS)**
A type of file system that resides in a Virtual Storage Access Method (VSAM) linear data set (LDS) and has a hierarchical organization of files and directories with a root directory.
Index

Special characters
\ (backslash) xi
# (pound sign) xi

A
abort command 99, 109
accessibility 331
ACL (access control lists) 3
active file system 20
adding a volume 31
address space 19
determining usage 89
aggregates
version 1.5 24
aggregate 110, 111
adding a volume 31
backup 63
changing, of a mounted zFS read/write file system 17
changing, of a mounted zFS read/write file system 17
cache debugging 89
correction, namespace 103
corruption, file system 99
creating
compatibility mode aggregate
adding a volume 31
compatibility mode aggregate
adding a volume 31
compatibility mode file system 21
maximum size 44
minimum size 44
mounting 23
configuration operations 208
configuration
zFS (z/OS File System) 11
compatibility
zFS file system 21
creating
compatibility mode file system 21
zFS file system 21
compatibility
zFS file system 21
considerations
symplex 51
contention, lock 85
correction 27
converting auditfids 115
correction, namespace 103
corruption, file system 99
creating
compatibility mode file system 21
zFS file system 21
CTKC report 75

C
cache
debugging 89
log file 72
metadata 70
transaction 71
user file 71
cache report
VM 94
cache size
IOEFSPRM 69
storage shortage 109
total 69
catch-up mount 4, 52
correction, namespace 103
corruption, file system 99
creating
compatibility mode file system 21
zFS file system 21
commands
continued
zfsadm grow 28, 180
zfsadm help 181
zfsadm lsaggr 183
zfsadm lsfs 184
zfsadm lsquota 186
zfsadm query 186
zfsadm setauditfid 188, 189
zfsadm unquiesce 191
compatibility mode aggregate
adding a volume 31
changing attributes 38
creating 29
decreasing size of 36
deleting 37
disabled 110
dynamically growing 28
growing 27
increasing size of 31
renaming 37
size 44
compatibility mode file system 21
maximum size 44
minimum size 44
mounting 23
configuration operations 208
configuration
zFS (z/OS File System) 11
compatibility
zFS file system 21
creating
compatibility mode file system 21
zFS file system 21
compatibility
zFS file system 21
considerations
symplex 51
contention, lock 85
conversion 27
converting auditfids 115
correction, namespace 103
corruption, file system 99
creating
compatibility mode file system 21
zFS file system 21
CTKC report 75

D
data sets
IOEFSPRM 193
DATASET report 75
debugging
89
storage
89
storage shortage 109
define aggregate subcommand 213
definitions
anode 132
attach 4
catch-up mount 4
file system ownership 5
function shipping 6
local mount 6
non-symplex aware 6
read-only file system 6
read-write file system 6
shared file system environment 6
symplex 7
definitions (continued)
sysplex-aware 7
sysplex-aware file system 7
sysplex-aware PFS 7
z/OS UNIX file system owner 5
zFS aggregate 7
zFS file system 8
zFS file system owner 5
zFS physical file system 8
delay of mount 39
delay, troubleshooting 104
delete file system subcommand 216
detach aggregate subcommand 220
dfs_cptiles program 12
DFSMsdss logical dump using for backup 63
diagnosing disabled 110
no z/OS V1R13 or later systems 111
z/OS V1R13 or later 111
directory
creating 13
determining size 45
extended (v5) 24
size 45
directory space
how to reclaim 46
disabled aggregates 110, 111
compatibility mode aggregate 110
disk space allocation
understanding 40
.dumps 101
dynamically growing compatibility mode aggregates 28

e
ENQs, displaying 103
examples
creating compatibility mode file system 21
define aggregate 214
delete file system 218
detach aggregate 221
format aggregate 224, 272
grow aggregate 227
ioeagfmt 128
ioeagslv command 134
IOEFSPRM sample file 204
ioefsult converttov4 command 136
ioeagfmt converttov5 command 138
ioeagfmt format 141
ioeagfmt salvage command 146
list aggregate status 229
list aggregate status (version 2) 233
list attached aggregate names 239, 246
list attached aggregate names (version 2) 242
list file system names 250, 253
list systems 263
logical restore 65
modify zfs process 122
query config option 266
quiesce aggregate 270
replace 65
restore 64
set config option 273, 275
examples (continued)
setomvs reset 123
statistics directory cache
information 277
statistics iobyaagr information 281
statistics iobyaagd information 285
statistics iobyaagd information 289
statistics kernel information 292
statistics locking information 296
statistics log cache information 299
statistics metadata cache
information 303
statistics storage information 307
statistics transaction cache
information 311
statistics user cache information 314
statistics vnode cache
information 319
unquiesce aggregate 322
zfs aggregate restore 64
zfs back up 64
zfsadm aggrinfo command 154
zfsadm apropos command 156
zfsadm attach command 158
zfsadm config command 162
zfsadm configquery command 165
zfsadm convert command 167
zfsadm define command 170
zfsadm delete command 171
zfsadm grow command 181
zfsadm help command 182
zfsadm lsaggr command 183
zfsadm lsfs command 185
zfsadm quota command 186
zfsadm quiesce command 189
zfsadm setauditfid command 190
zfsadm unquiesce command 192
extended (v5) aggregate
converting to version 1.5 25
extended (v5) directories 24
extended director XCF communications protocol 102

f
Fast Response Cache Accelerator
restriction 15
features 3
zFS 3
file allocation
blocked 42, 70
fragmented 42, 70
inline 41, 70
FILE report 76
file system
corruption 99
determining owner 56
dynamic movement 59
maximum size 44
minimum size 44
ownership 56, 58
read-only
sysplex-aware 51
read/write
non-sysplex aware 52
sysplex-aware 53
status 20
file system (continued)
sysplex-aware 7
z/OS UNIX owner 58
file system operations 208
file system owner
z/OS UNIX 55
zFS 55
file system ownership 5
file system, zFS 8
files
IOEFSPRM 193
fixed storage 72
format aggregate subcommand 222
fragmented file allocation 42, 70
function shipping 6

G
grow aggregate subcommand 225
guidelines for v4 to v5 conversion 27

h
hang
detection 104
steps for resolving 105
health check 15, 17

i
I/O
balancing 73
statistics 77
initialization messages, saving in a data set 102
inline file allocation 41, 70
installing
zFS (z/OS File System) 11
intermediate archive file 68
internal restart 99, 109
IOBYDASD
related subcommand 283
report 77
ioefsult

disable aggregate 112
ioeagfmt command 125, 128
ioeagfmt utility 21
ioeagfmt command 129
commands
ioeagfmt 129
example 134
IOEFSPRM 193, 195
example 204
sharing 60
total cache size 69
ioefsult converttov4 command 135
commands
ioefsult converttov4 135
example 136
ioefsult converttov5 command 137
example 138
ioefsult format command 139, 141
ioefsutl salvage
  command 142
  example 146
ioefsutl utility 134
IOEPRMxx 195
IOEZADM module 152

J
JES 12

K
keyboard
  navigation 331
  PF keys 331
  shortcut keys 331
KN report 78

L
large directory 46
largedir.pl command 45
LFS report 79
list aggregate status (version 2)
  subcommand 230
list aggregate status subcommand 228
list attached aggregate name
  subcommand 238
list attached aggregate names (version 2)
  subcommand 241
List File Information subcommand 244
list file system names subcommand 249, 252
list file system status subcommand 255
list systems subcommand 263
local mount 6, 52
LOCK report 85
log file cache 72
log files 72
LOG report 88

M
man pages
  enabling 125
  example of command 125
managing
  processes 19
  zFS file system 21
maximum size
  for file system 44
messages 195, 203
messages, initialization, saving in a data
  set 102
metadata cache 70
migrating
  from HFS to zFS 67
  using the z/OS HFS to zFS migration tool 67
minimum size
  for file system 44
modify zfs process command 119
  examples 122
monitoring performance periods 74
mount command 23
MOUNT command 146
mount wait 39
mount, local 6
mounting
  compatibility mode file system 23
  same mode 112
msg_output_dsn option of
  IOEFSPRM 102
multilevel security 3

N
namespace validation 103
namespace validation 331
navigation
  keyboard 331
NBS (New Block Security) 157
New Block Security (NBS) 157
NLS 195, 203
no z/OS VIR13 or later systems 111
non-sysplex aware 6, 52
NORWSHARE
  MOUNT 148
  zfsadm config 161
Notices 335

O
objects
  maximum number 44
  obtaining 101
options
  zFS PFS 193
  owner of file system 55
  owner, file system 5

P
path entry 148
pax command 67
performance 4
  number of file names 45
  performance considerations 69
performance, monitoring 73
PFS (physical file system) 8, 193
state 20
sysplex-aware 7
pfsctl 207
aggregate operations 208
configuration operations 208
file system operations 208
query operations 208
physical file system (PFS) 8
Policy Agent Server (Pagent)
  restriction 15
post installation processing 11

Q
query config option subcommand 265
query operations 208
QUERY,KN report 78
QUERY,STOR 89
quiesce aggregate subcommand 269
quota 22

R
RACF commands 12
read-only
  file system 51
read-only file system 6
read/write
  file system 52
read/write file system 6
reason codes, using bpxmtext 125
reset backup flag subcommand 271
resetting performance data 74
resolving hangs 105
restart 4
restart, internal 109
restore
  from back up 64, 65
root, large directory 46
RWSHARE
  MOUNT 148
  zfsadm config 161

S
salvager utility 99, 129, 142
same mode 112
scan for small files 40
security label 3
sending comments to IBM xiii
service level, determining 102
set auditfid subcommand 273
set config option subcommand 274
setomvs reset command 122
  examples 123
shared file system 51
  overview 51
shared file system environment 6
z/OS UNIX consideration 60
sharing zfs data between systems 43
shortcut keys 331
small files, scanning 40
SMB
  effect on user file cache hit ratio 72
  restriction 15
running dfs_cpfiler program 12
SMF record
  auditid 113
specifying 15
specifying as sysplex-aware 15
statistics directory cache information
  subcommand 276
statistics 1obyaggr information
  subcommand 279
statistics 1obydasd information
  subcommand 283
statistics 1oinuxt information
  subcommand 288
statistics kernel information
  subcommand 291
statistics locking information
  subcommand 294
statistics log cache information
  subcommand 298
statistics metadata cache information
  subcommand 301
statistics storage information
  subcommand 305

Index 343
zfsadm setauditfid command  (continued)
  example  190
zfsadm unquiesce command
  example  192
zfsspace utility  40