Distributed File Service
zSeries File System
Administration
Distributed File Service
zSeries File System
Administration
Contents

Figures ........................................................................ v i i

About this document ................................................................. ix
How this document is organized .................................................. ix
Conventions used in this document ............................................. ix
Where to find more information ................................................ x
  Softcopy publications ......................................................... x
  Internet sources ................................................................ x
  Information updates on the web ............................................... x
  The z/OS Basic Skills Information Center ............................... x

Summary of Changes ................................................................ xi

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

Chapter 1. zSeries File System (zFS) overview ............................... 3
  Features ........................................................................ 3
  Terminology and concepts .................................................. 4
  What’s new or changed for zFS in z/OS V1R11 ......................... 7

Chapter 2. zFS post installation processing ................................. 11
  Two-step APAR procedure for z/OS V1R11 ......................... 11
  Enabling the zFS sysplex-aware function ............................ 12
    - Running all systems with zFS non-sysplex aware ............... 13
    - Running all systems with zFS sysplex-aware .................. 13
    - Running mixed zFS sysplex-aware and zFS non-sysplex aware .... 13
  zFS installation and configuration steps .............................. 14

Chapter 3. Managing zFS processes .......................................... 17
  Starting zFS ................................................................. 17
  Stopping zFS ............................................................... 17
    - Determining file system status ................................... 18

Chapter 4. Creating and managing zFS file systems using compatibility mode aggregates .... 19
  Creating a compatibility mode aggregate ......................... 19
  Growing a compatibility mode aggregate ......................... 21
  Dynamically growing a compatibility mode aggregate ........ 21
  Creating a multi-volume compatibility mode aggregate ....... 22
  Adding a volume to a compatibility mode aggregate .......... 22
  Renaming or deleting a compatibility mode aggregate ....... 23
  Unmounting zFS file systems before mounting, copying or backing up .................... 23
  Cloning a file system ..................................................... 24
  zFS disk space allocation ............................................... 25
  Sharing zFS data in a non-shared file system sysplex ............ 27
  Minimum and maximum file system sizes ......................... 28

Chapter 5. Using zFS in a shared file system environment ............. 31
  Overview of the shared file system environment ................. 31
  Read-only mounted file systems ....................................... 31
  zFS support for read/write non-syplex aware mounted file systems ....... 32
  zFS support for read/write sysplex-aware mounted file systems .... 33
  zFS ownership versus z/OS UNIX ownership of file systems ....... 34
    - Determining the file system owner .......................... 35

© Copyright IBM Corp. 2001, 2009
Chapter 6. Performing a backup of zFS

Back up a zFS aggregate

Restoring an aggregate with DFSMSdss logical restore

Chapter 7. Migrating data from HFS to zFS

Using the z/OS HFS to zFS migration tool

Using the z/OS UNIX pax command

Using an intermediate archive file

Without using an intermediate archive file

Chapter 8. Multi-file system aggregates

Creating a multi-file system aggregate

Growing a multi-file system aggregate

Dynamically growing a multi-file system aggregate

When an aggregate or file system becomes full

Comparing compatibility mode aggregates and multi-file system aggregates

Sharing zFS data between systems

Chapter 9. Performance and debugging

Performance tuning

Total cache size

Metadata cache

Transaction cache

Vnode cache

User file and client file cache

NOREADAHEAD option

Log files

Log file cache

Fixed storage

I/O balancing

Monitoring zFS performance

Sample zFS query reports

Debugging aids for zFS

Overview of trace options for zFS

Understanding the salvager utility

Overview of dumping for zFS

Understanding zFS messages

Determining service levels

Understanding namespace validation and correction

Understanding delays and hangs in zFS using the zFS hang detector

Storage shortage in zFS

Diagnosing disabled aggregates

Disabled compatibility mode aggregate

Disabled multi-file system aggregate

Chapter 10. Overview of the zFS audit identifier

Enabling the zFS auditid
Part 2. zFS administration reference ................................................. 93

Chapter 11. z/OS system commands .................................................. 95
modify zfs process ........................................................................... 96
setomvs reset .................................................................................. 99

Chapter 12. zFS commands ............................................................... 101
ioeagfmt ........................................................................................ 102
ioeagslv ......................................................................................... 105
MOUNT .......................................................................................... 109
zfsadm ............................................................................................ 113
zfsadm aggrinfo .............................................................................. 117
zfsadm apropos .............................................................................. 120
zfsadm attach ............................................................................... 121
zfsadm clone ................................................................................. 124
zfsadm clonesys ............................................................................ 126
zfsadm config ............................................................................... 128
zfsadm configquery ...................................................................... 131
zfsadm create ............................................................................... 134
zfsadm define ............................................................................... 137
zfsadm delete ............................................................................... 139
zfsadm detach ............................................................................... 141
zfsadm format ............................................................................... 143
zfsadm grow .................................................................................. 145
zfsadm help ................................................................................. 147
zfsadm isaggr ............................................................................... 148
zfsadm lsfs ................................................................................... 149
zfsadm lsquota ............................................................................. 151
zfsadm lssys .................................................................................. 153
zfsadm query ............................................................................... 154
zfsadm quiesce ............................................................................ 156
zfsadm rename ............................................................................. 158
zfsadm setauditfid ...................................................................... 160
zfsadm setquota .......................................................................... 161
zfsadm unquiesce ........................................................................ 163

Chapter 13. zFS data sets ................................................................. 165
IOEFSPRM .................................................................................... 166

Chapter 14. zFS application programming interfaces ....................... 177
pfsect (BPX1PCT) .......................................................................... 178
Attach Aggregate ......................................................................... 182
Clone File System ......................................................................... 186
Create File System ....................................................................... 191
Define Aggregate ......................................................................... 197
Delete File System ...................................................................... 201
Detach Aggregate ......................................................................... 206
Format Aggregate ......................................................................... 209
Grow Aggregate ........................................................................... 213
List Aggregate Status .................................................................. 216
List Aggregate Status (Version 2) .................................................. 220
List Attached Aggregate Names .................................................... 224
List Attached Aggregate Names (Version 2) .................................... 228
List File System Names .................................................................. 232
List File System Names (Version 2) ................................................ 236
List File System Status ................................................................ 240

Contents  V
Figures

1. z/OS UNIX and zFS file system ownership ........................................... 5
2. Job to create a compatibility mode file system .................................. 19
3. Job to create a multi-volume compatibility mode aggregate .................. 22
4. Disk space allocation example 1 ....................................................... 26
5. Disk space allocation example 2 ....................................................... 26
6. Sysplex-aware file system (read-only) ............................................... 32
7. Non-sysplex aware file system (read/write) ....................................... 33
8. zFS sysplex-aware for read/write mounted file systems ......................... 34
9. zFS sysplex-aware with new owner .................................................. 34
10. zfsadm lsaggr and df -v output after mount ..................................... 35
11. zfsadm lsaggr and df -v output after movement ................................ 35
12. File system ownership when mount fails ......................................... 36
13. One system non-sysplex aware ....................................................... 37
14. Non-sysplex aware with different owners ........................................ 37
15. zFS non-sysplex aware system with sysplex-aware systems ................. 38
16. Steps for quiesce and unquiesce ..................................................... 43
17. Job to back up a zFS aggregate ..................................................... 44
18. Job to restore a zFS aggregate ..................................................... 44
19. Job to restore a zFS aggregate with replace .................................... 45
20. Job to create a multi-file system aggregate ..................................... 50
21. zFS auditid examples ..................................................................... 89
22. Job to create a compatibility mode aggregate and file system ............. 104
23. Job to verify a zFS aggregate ....................................................... 108
24. Job to display aggregate information .............................................. 119
25. Job to attach an aggregate ............................................................ 123
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 zSeries 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 discusses guidance information for the zSeries File System (zFS).
• Part 2, “zFS administration reference,” on page 93 discusses the zSeries File System (zFS) reference information 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** 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 non-blank 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).

This document used the following keying convention:

<Return> The notation <Return> refers to the key on your terminal or workstation that is labeled with either the word “Return” or “Enter”, with a left arrow.

**Entering commands**

When instructed to enter a command, type the command name and then press <Return>. 
Where to find more information

Where necessary, this document references information in other documents. For complete titles and order numbers for all elements of z/OS, refer to the [z/OS Information Roadmap](http://www.ibm.com/support/docview.wss?rs=132&uid=swg27013285).

Information about installing Distributed File Service components is found in [z/OS Program Directory](http://www.ibm.com/support/docview.wss?rs=132&uid=swg27013285).

Information about Distributed File Service zSeries File System-related messages is found in [z/OS Distributed File Service Messages and Codes](http://www.ibm.com/support/docview.wss?rs=132&uid=swg27013285).

Softcopy publications

The z/OS Distributed File Service library is available on a CD-ROM, z/OS Collection, SK3T-4269. The CD-ROM online library collections is a set of documents for z/OS and related products that includes the IBM Library Reader. This is a program that enables you to view the BookManager® files. This CD-ROM also contains the Portable Document Format (PDF) files. You can view or print these files with the Adobe Acrobat reader.

Internet sources

The softcopy z/OS publications are also available for web-browsing and for viewing or printing PDFs using the following URL: [http://www.ibm.com/systems/z/os/zos/bkserv/e0zlib](http://www.ibm.com/systems/z/os/zos/bkserv/e0zlib).

You can also provide comments about this document and any other z/OS documentation by visiting that URL. Your feedback is important in helping to provide the most accurate and high-quality information.

Information updates on the web

For the latest information updates that have been provided in PTF cover letters and Documentation APARs for z/OS, see the online document at: [http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/ZDOCAPAR](http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/ZDOCAPAR).

This document is updated weekly and lists documentation changes before they are incorporated into z/OS publications.

The z/OS Basic Skills Information Center

The z/OS Basic Skills Information Center is a Web-based information resource intended to help users learn the basic concepts of z/OS, the operating system that runs most of the IBM mainframe computers in use today. The Information Center is designed to introduce a new generation of Information Technology professionals to basic concepts and help them prepare for a career as a z/OS professional, such as a z/OS system programmer.

Specifically, the z/OS Basic Skills Information Center is intended to achieve the following objectives:

- Provide basic education and information about z/OS without charge
- Shorten the time it takes for people to become productive on the mainframe
- Make it easier for new people to learn z/OS.

To access the z/OS Basic Skills Information Center, open your Web browser to the following Web site, which is available to all users (no login required): [http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp](http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp)
Summary of Changes

Summary of changes for SC24-5989-10
z/OS Version 1 Release 11

New information

- "The z/OS Basic Skills Information Center" on page x.
- You can now enable zFS to run in sysplex-aware mode. Even if your installation is not sysplex-aware, you must follow the appropriate migration actions. See "Enabling the zFS sysplex-aware function" on page 12 and "Chapter 5, "Using zFS in a shared file system environment," on page 31 for complete details. The following information also contains information about sysplex-aware zFS:
  - Chapter 9, "Performance and debugging," on page 57
  - Chapter 11, "z/OS system commands," on page 95
  - Chapter 12, "zFS commands," on page 101
  - "IOEFSPRM" on page 166
  - Chapter 14, "zFS application programming interfaces," on page 177.
- New information about quiescing before backing up zFS, see Chapter 6, "Performing a back up of zFS," on page 43.
- New consideration when migrating your sysplex root, see Chapter 7, "Migrating data from HFS to zFS," on page 47.
- You cannot attach a zFS multi-file system aggregate in a sysplex shared file system environment. For more details, see Chapter 8, "Multi-file system aggregates," on page 49.
- zFS introduces the concept of namespace validation and correction, see "Understanding namespace validation and correction" on page 78.
- New information about the salvager (ioeagslv) utility, see "Understanding the salvager utility" on page 75.
- This documentation now contains a glossary, see "Glossary" on page 353.

Changed information

- "Stopping zFS" on page 17 contains information about automatic ownership movement.
- Chapter 9, "Performance and debugging," on page 57 contains updated information and examples that support zFS sysplex-aware.
- The -vnode_cache_limit is ignored. Information pertaining to the -vnode_cache_limit has been removed from this documentation.
- The following APIs have an additional restriction:
  - "Clone File System" on page 186
  - "Create File System" on page 191
  - "Delete File System" on page 201
  - "List File System Names (Version 2)" on page 236
  - "List File System Status" on page 240
  - "Rename File System" on page 261
  - "Set File System Quota" on page 273
- Clarified information that zfsadm commands can be issued from TSO/E using the IOEZADM format. See "zfsadm" on page 113.
You may notice changes in the style and structure of some content in this document—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

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

Summary of changes for SC24-5989-09
z/OS Version 1 Release 10

New information

• Information is added to clarify the maximum number of objects allowed in a zFS file system. For details, see "Minimum and maximum file system sizes" on page 28.

• You can now use the man command to view the descriptions of zFS command manual pages. For details, see Chapter 12, “zFS commands,” on page 101.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Summary of changes for SC24-5989-08
z/OS Version 1 Release 9
as updated April 2008

This document contains information previously presented in SC24-5989-07, which supports z/OS Version 1 Release 9.

New information

• An important guideline about "Unmounting zFS file systems before mounting, copying or backing up" on page 23.

• With APAR OA20614 applied to z/OS V1R9, zFS allows you to specify whether zFS is to use a new, more unique auditid for a zFS file or directory, or to continue using the existing (non-unique) auditid. See Chapter 10, “Overview of the zFS audit identifier,” on page 89 and "zfsadm setauditfid" on page 160.

• With APAR OA20613 applied to z/OS V1R9, zFS allows users with ALTER authority to format a zFS aggregate. See "ioeagfmt" on page 102 and "ioeagslv" on page 105.

• With APAR OA20615 applied to z/OS V1R9, zFS allows log recovery for read-only mounts. See "zfsadm config" on page 128, "zfsadm configquery" on page 131, and "romount_recovery" on page 170 in "IOEFSPRM" on page 166.

Changed information

• The Create File System API fd_quota field is changed from long to unsigned long. See "Create File System" on page 191 for details.

• To make them easier to locate, zFS command options are now in alphabetic order. See Chapter 12, "zFS commands," on page 101.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Summary of changes for SC24-5989-07
z/OS Version 1 Release 9
This document contains information previously presented in SC24-5989-06, which supports z/OS Version 1 Release 8.

**Changed information**
- The default for the IOEFSRM `dir_cache_size` option is now 32M. See “IOEFSRM” on page 166.
- Information about “Cloning a file system” on page 24 has been moved to Chapter 4, “Creating and managing zFS file systems using compatibility mode aggregates,” on page 19.
- **MODIFY ZFS,QUERY** has new report options. See “Monitoring zFS performance” on page 60.

**Deleted information**
- The `zfsadm configquery` option `-allow_dup_fs` and the `zfsadm config` option `-allow_dup_fs` are removed.
- The IOEFSRM `allow_duplicate_filesystems` option is no longer optional. As of z/OS V1R9, the condition is always `allow_duplicate_filesystems=on`.
- z/OS.e is not supported in z/OS V1R9.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.
Part 1. zFS administration guide

This part of the document discusses guidance information for the zSeries File System (zFS).

- Chapter 1, “zSeries File System (zFS) overview,” on page 3
- Chapter 2, “zFS post installation processing,” on page 11
- Chapter 3, “Managing zFS processes,” on page 17
- Chapter 4, “Creating and managing zFS file systems using compatibility mode aggregates,” on page 19
- Chapter 5, “Using zFS in a shared file system environment,” on page 31
- Chapter 6, “Performing a back up of zFS,” on page 43
- Chapter 7, “Migrating data from HFS to zFS,” on page 47
- Chapter 8, “Multi-file system aggregates,” on page 49
- Chapter 9, “Performance and debugging,” on page 57
Chapter 1. zSeries File System (zFS) overview

The z/OS® Distributed File Service zSeries 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, GA22-7800.

zFS does not replace HFS, rather zFS is complementary to HFS. zFS can be used for all levels of the z/OS UNIX System Services hierarchy (including the root file system) when all members are at the z/OS V1R7 level. Because zFS has higher performance characteristics than HFS and is the strategic file system, HFS might not be supported in future releases, which will cause you to migrate the remaining HFS file systems to zFS.

Beginning in z/OS V1R11, zFS has the capability of running sysplex aware for read-write mounted file systems. As in previous releases, zFS continues to support running sysplex-aware for read-only mounted file systems. For more information, see “Enabling the zFS sysplex-aware function” on page 12, “Terminology and concepts” on page 4, and Chapter 5, “Using zFS in a shared file system environment,” on page 31.

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, GA22-7509 for more information about multilevel security and migrating your HFS version root to a zFS version root with security labels.

Note: Multi-file system aggregate support is not planned to be enhanced and might be removed sometime in the future. IBM recommends that you use zFS compatibility mode aggregates rather than zFS multi-file system aggregates. If you have any 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.
- Beginning with z/OS V1R8, zFS file systems that are located in a zFS multi-file system aggregate in a shared file system environment cannot be mounted.
- Beginning with z/OS V1R11, zFS multi-file system aggregates cannot be attached in a shared file system environment. 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.

Features

zFS provides many features and benefits:

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 in sysplex-aware mode, zFS automatically moves zFS ownership of a zFS file system to the system that has the most read-write activity. “Terminology and concepts” on page 4 has an
Cloning

As an optional function, zFS allows the administrator to make a read-only clone of a file system in the same data set. This clone file system can be made available to users to provide a read-only point-in-time copy of a file system. The clone operation happens relatively quickly and does not take up too much additional space because only the metadata is copied.

Note: This function has some restrictions. For information about these restrictions, see "Cloning a file system" on page 24.

Terminology and concepts

In order to present all the benefits and details of zFS administration, the following new 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. For example, if a zFS clone (backup) file system is mounted (read-only) after the read-write file system has been mounted, an attach does not occur on the second mount of a file system contained in the data set. The data set is already attached. MOUNT PARMs for compatibility mode file systems only take effect on the first MOUNT (that does the attach). Conversely, a detach is not done until both file systems in a particular data set are unmounted.

Backup file system

The backup file system is the result of a zFS clone operation. The backup file system is a read-only file system and can only be mounted as read-only.

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, the catch-up mount, to every other system’s PFS that is running sysplex-aware for that mode (read-write or read-only). 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. 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.
**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. Having the appropriate owner is important for performance. We use the term **z/OS UNIX file system owner** to mean 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](#secZFSOwner)).

**zFS file system owner**

zFS has its own concept of file system ownership. We call this owner 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](#fig1). 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.  

zFS read/write mounted file system running zFS non-sysplex aware  
(Releases prior to z/OS V1R11 and V1R11 with sysplex=off)  

<table>
<thead>
<tr>
<th>z/OS UNIX application</th>
<th>z/OS UNIX</th>
<th>z/OS UNIX</th>
<th>z/OS UNIX</th>
<th>z/OS UNIX</th>
<th>z/OS UNIX</th>
<th>z/OS UNIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS UNIX</td>
<td>z/OS UNIX</td>
<td>z/OS UNIX</td>
<td>z/OS UNIX</td>
<td>z/OS UNIX</td>
<td>z/OS UNIX</td>
<td>z/OS UNIX</td>
</tr>
<tr>
<td>zFS</td>
<td>zFS</td>
<td>zFS</td>
<td>zFS</td>
<td>zFS</td>
<td>zFS</td>
<td>zFS</td>
</tr>
</tbody>
</table>

zFS read/write mounted file system running zFS sysplex-aware  
(Releases beginning with z/OS V1R11 with sysplex=on)  

<table>
<thead>
<tr>
<th>z/OS UNIX application</th>
<th>z/OS UNIX</th>
<th>z/OS UNIX</th>
<th>z/OS UNIX</th>
<th>z/OS UNIX</th>
<th>z/OS UNIX</th>
<th>z/OS UNIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS UNIX</td>
<td>z/OS UNIX</td>
<td>z/OS UNIX</td>
<td>z/OS UNIX</td>
<td>z/OS UNIX</td>
<td>z/OS UNIX</td>
<td>z/OS UNIX</td>
</tr>
<tr>
<td>zFS</td>
<td>zFS</td>
<td>zFS</td>
<td>zFS</td>
<td>zFS</td>
<td>zFS</td>
<td>zFS</td>
</tr>
</tbody>
</table>

![Figure 1. z/OS UNIX and zFS file system ownership](#fig1)

**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”](#detFSOwner) on page 35 and Chapter 5, “Using zFS in a shared file system environment,” on page 31.

**Function shipping**

Function shipping means that a request is forwarded to the owning system and the response is returned back to the requestor through XCF communications.

---

2. In reality, zFS owns aggregates. Generally, we simplify this to say zFS file system owner because, in most cases, zFS compatibility mode aggregates only have a single file system.
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 PFS is running 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 35.

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. In non-sysplex aware zFS, 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” on page 7 and “z/OS UNIX file system owner” on page 5.

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

When this documentation refers to the term sysplex as it applies to zFS, it 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

A PFS (Physical File System) that is sysplex-aware for a particular mode (read-only or read-write) allows file requests for file systems that are mounted in that mode to be handled by that local PFS (that is, they are locally mounted on that system). z/OS UNIX does not function ship those file requests to the z/OS UNIX owning system. Note that if the file system volume is not online to a system, then the system becomes a client to the file system through function shipping with the owner because the local mount is unsuccessful. Beginning with z/OS V1R11, you can enable zFS to be sysplex-aware for read-write mounted file systems. Conversely to sysplex-aware, there also exists the concept of non-sysplex aware. See “Non-sysplex aware (sysplex-unaware).”

zFS aggregate

The data set that contains zFS file systems is called a zFS aggregate. A zFS aggregate can contain one or more zFS file systems. A zFS aggregate is a Virtual Storage Access Method Linear Data Set (VSAM LDS). After the zFS aggregate is defined and formatted, one or more zFS file systems can be created in the aggregate. A zFS aggregate that contains only a single read-write zFS file system can be defined and is called a compatibility mode aggregate. A compatibility mode aggregate can also contain a backup file system. Compatibility mode aggregates are similar to HFS.

Note: It is recommended that as you begin to use zFS, you use compatibility mode aggregates. Aggregates that contain multiple file systems are called multi-file system aggregates. This support is not planned to be enhanced and might be removed in the future.

3. A zFS aggregate can contain zero or more zFS file systems.
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 LDS, it will only mount as read-only. zFS does not support a zFS aggregate that has guaranteed space.

zFS file system

The term 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 file system name

zFS file system name is the name of the file system as zFS knows it. The term z/OS UNIX file system name or mount file system name refers to the name of the file system as z/OS UNIX knows it. We make this distinction because you can now specify a z/OS UNIX file system name (as specified in the MOUNT FILESYSTEM option) that is different from the zFS file system name (as optionally specified in the MOUNT PARM FILESYSTEM suboption). This latter specification might be required when working with multiple zFS file systems that have the same zFS file system name (in different zFS aggregates). However, this capability is only useful for multi-file system aggregates. There is no plan for the multi-file system aggregate function to be enhanced and it might be removed in the future. Therefore, do not use this capability. If you are currently using multi-file system aggregates, start planning to discontinue.

zFS Physical File System (PFS)

The term 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 V1R11

Beginning with z/OS V1R11, you can enable zFS to run sysplex-aware. For details, see “Enabling the zFS sysplex-aware function” on page 12. Also see the chapter on Chapter 5, “Using zFS in a shared file system environment,” on page 31, which includes “Overview of the shared file system environment” on page 31 and additional topics on:

• “Read-only mounted file systems” on page 31
• “zFS support for read/write non-syplex aware mounted file systems” on page 32
• “zFS support for read/write sysplex-aware mounted file systems” on page 33
• “zFS ownership versus z/OS UNIX ownership of file systems” on page 34
• “Determining the file system owner” on page 35
• “When is the z/OS UNIX owner important?” on page 36
• “What happens when one system runs zFS non-syplex aware?” on page 36
• “What happens when a zFS non-syplex aware system owns a file system and the other systems are running sysplex-aware?” on page 37

Restriction for z/OS Distributed File Service Server Message Block (SMB):

The SMB server cannot export zFS read-write file systems when zFS is running sysplex-aware on the same system the SMB server is running or on the system that owns the zFS file system. To export zFS file systems using the SMB server, you must run zFS non-syplex aware by specifying sysplex=off in the IOEFSPRM.

Restriction for IBM HTTP Server Version 5.3 for z/OS when using Fast Response Cache Accelerator:

When the IBM HTTP Server Version 5.3 for z/OS is configured to use Fast Response Cache Accelerator, static web pages are cached to improve performance. Part of this support uses register file interest, a function of z/OS UNIX, to get notified if the files representing the web pages are changed.
When you specify sysplex=off in the IOEFS(PR)M configuration file, z/OS UNIX sysplex file sharing is performed at the z/OS UNIX layer. In this mode, when a file (for which register file interest is active) changes, z/OS UNIX is able to notify the HTTP Server of the change. When you specify sysplex=on, z/OS UNIX file sharing is performed at the zFS layer. In this mode, attempts to register file interest in a zFS read/write file system are rejected, which means the HTTP Server cannot cache pages and performance will degrade. The HTTP Server continues to work, but the pages sent to clients are not cached.

Use the following procedure if you must cache files for the static web pages when zFS has sysplex=on and is in a zFS read/write file system:

1. Copy the files to an HFS file system or to a zFS file system that is mounted read-only mode when IBM HTTP Server uses it.
2. Change your settings in IBM HTTP Server Version 5.3 for z/OS with Fast Response Cache Accelerator to load the files from the new file system.

Additional considerations when deciding to run zFS sysplex-aware:

- Before the first IPL of V1R11, if you have z/OS V1R9, V1R10, or both releases in your sysplex, you must follow the zFS migration actions as described in [z/OS Migration, GA22-7499](#).
- If you are running zFS and you are running other sysplex members at previous releases, you must ensure that you have (at least) APAR OA25026 installed on the prior releases. See [“Two-step APAR procedure for z/OS V1R11” on page 11](#) for additional details.
- When the BPXPRMxx parmlib member specifies SYSPLEX(YES), you can specify sysplex=on in the IOEFS(PR)M configuration file and perform an IPL. After changing IOEFS(PR)M, you must perform an IPL or a restart of zFS. IPL is the recommended method because in many ways it is less disruptive than a restart of zFS, which can cause zFS file systems to become unmounted. This causes zFS to run sysplex-aware and z/OS UNIX does not forward file requests to the z/OS UNIX owning system. Rather, z/OS UNIX typically sends requests to the local zFS and lets zFS take the responsibility to forward the request, if necessary. (For information about when z/OS UNIX function ships requests to the z/OS UNIX owner even when zFS is running sysplex-aware, see “When is the z/OS UNIX owner important?” on page 36.) Cache consistency is maintained through a token management mechanism. This typically provides improved performance in a shared file system environment.
- When zFS is running sysplex-aware for read-write file systems, a new cache is added to zFS controlled by the `client_cache_size` IOEFS(PR)M configuration option. The `client_cache_size` option is used to cache user data when zFS files are accessed from a system that is not the zFS owner of the file system. The default size is 128 MB. This is in addition to the existing `user_cache_size` option, which is used when zFS files are accessed from a system that is the zFS owner of the file system. For information about tuning these caches, see “Monitoring zFS performance” on page 60.
- When zFS is running sysplex-aware for read-write file systems, zFS can move the zFS ownership of an aggregate based on one of the following conditions:
  - the number of I/O requests from each system
  - the failure of the zFS owning system
  - the stop or end of zFS.

As a result, the zFS owner of the aggregate can be different than the z/OS UNIX owner of the file system. To display the z/OS UNIX owner, use `df -v` command; to display the zFS owner, use the `zfsadm lsaggr` command.

- In a shared file system environment, after a zFS file system that is mounted read-write becomes z/OS UNIX owned by a z/OS V1R11 system running zFS sysplex-aware, in general, it cannot be explicitly moved by z/OS UNIX to a prior release system or to a z/OS V1R11 system that is running zFS non-sysplex aware. In the special case that the z/OS V1R11 system running zFS sysplex-aware is the z/OS UNIX owning system and every other system is function shipping requests to this system, the zFS read-write file system can be explicitly moved, in most cases.
In a shared file system environment, when a zFS file system is mounted read-write and is z/OS UNIX owned on a prior release (or is z/OS UNIX owned on a z/OS V1R11 system that is running zFS non-sysplex aware), a message similar to BPXF221I is issued on each system that is running zFS sysplex-aware:

BPXF221I FILE SYSTEM PLEX.JMS.AGGR006.LDS0006 FAILED TO MOUNT LOCALLY.
RETURN CODE = 00000079, REASON CODE = EF0969A8
THE FILE SYSTEM IS ACCESSIBLE ON THIS SYSTEM THROUGH A MOUNT ON A REMOTE SYSTEM.

In this case, z/OS UNIX does forward file requests from the z/OS V1R11 system to the z/OS UNIX owning system.

**Tip:** IBM Health Checker for z/OS supplies the USS_CLIENT_MOUNTS check, which generates an exception message when it finds a file system that is function shipping, but can be mounted locally. See the topic on [USS_CLIENT_MOUNTS](https://www.ibm.com/docs/en/ibm-health-checker-for-zos/3.0?topic=check-uss-client-mounts) in IBM Health Checker for z/OS: User's Guide.

- The zFS XCF protocol is simplified. See “Understanding namespace validation and correction” on page 78.
- zFS multi-file system aggregates cannot be attached in a shared file system environment.

**What changed in zFS for V1R11:**

Beginning in V1R11, the following items apply:

- For compatibility mode aggregates, zFS ignores the FSGROW parameter on the MOUNT command.
- The -vnode_cache_limit setting is no longer a valid option. You can still set and query the value, but zFS processing ignores the setting.
- zFS supports a file format value of eight (RECORD format). This support is also available for z/OS V1R8 through V1R10 by applying APAR OA27728.
Chapter 2. zFS post installation processing

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 DCE DFS™ support nor the SMB server support of the Distributed File Service), DCE DFS and SMB do not need to be configured and DCE does not need to be configured. For more information about DCE DFS, see [SC24-5913](#). For more information about SMB, see [z/OS Distributed File Service SMB Administration, SC24-5918](#).

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 166.
- 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 BPXPRMxx member to cause zFS file systems to be mounted at IPL.

**Two-step APAR procedure for z/OS V1R11**

In z/OS V1R11, in addition to the "zFS installation and configuration steps" on page 14, you must also apply zFS sysplex administration function using the following two-step APAR procedure:

1. Install APAR OA25026 on all z/OS V1R9 and z/OS V1R10 systems. This is a conditioning function for zFS on z/OS V1R11. Make APAR OA25026 active on all systems through a rolling IPL. You are now running with zFS sysplex_admin_level=1.

2. After APAR OA25026 is active on each z/OS V1R9 and V1R10 system, specify the sysplex_admin_level=2 configuration option in the IOEFSPRM file. Make this level active on all z/OS V1R9 and V1R10 systems through another rolling IPL. This is the toleration function for zFS on z/OS V1R11. (The default for sysplex_admin_level is sysplex_admin_level=1). For information about specifying zFS configuration options, see Chapter 13, "zFS data sets," on page 165.

3. You cannot skip step 1 and do only step 2 if you are running any z/OS V1R9 or V1R10 systems that do not have APAR OA25026 installed and active.

**Notes:**

1. Your systems must be running zFS V1R9 and above to support a zFS V1R11 environment. In other words, zFS V1R11 is not compatible with V1R8 and below.

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

When sysplex_admin_level=1 or =2, the MODIFY ZFS,QUERY,LEVEL operator command returns a new line at the end of the output showing the sysplex admin interface level. The following example shows z/OS V1R10 displaying interface level 1.

```
MODIFY ZFS,QUERY,LEVEL
```

```
IOEZ00639I zFS kernel: z/OS zSeries File System
Version 01.10.00 Service Level 0Annnee - HZFS3A0.
Created on Tue Jan 6 20:02:25 EST 2009.
sysplex(admin-only) interface(1)
```
If you encounter a problem with a z/OS V1R9 or V1R10 system running with zFS sysplex_admin_level=2, you can remove the specification, or specify sysplex_admin_level=1 (this is equivalent to the default), and perform a rolling IPL or restart zFS on each system. This cannot be done after a zFS on the V1R11 system has joined the sysplex. Also, if you try to start zFS on z/OS V1R11 on another system after you have backed out to sysplex_admin_level=1 on zFS V1R9 or V1R10, zFS on z/OS V1R11 will not start because it requires all other systems to be at zFS sysplex_admin_level=2.

Tip: You need level sysplex_admin_level=2 on prior releases to bring up zFS on z/OS V1R11. Use the zFS migration check, ZOSMIGV1R11_ZFS_INTERFACELEVEL, in the IBM Health Checker for z/OS to help determine whether you are running zFS at the correct interface level. This check is automatically installed with APAR OA27198. See [IBM Health Checker for z/OS: User’s Guide, SA22-7994](#).

### Enabling the zFS sysplex-aware function

This section helps you determine whether to enable the zFS sysplex-aware function for zFS read/write file systems in a shared file system environment.

If you are running your sysplex in a shared file system environment, that is, BPXPRMxx specifies SYSPLEX(YES), you can set zFS V1R11 to run in either of the following modes:

- non-sysplex aware for read/write zFS file systems as in previous releases (the default)
- sysplex-aware for read/write zFS file systems (enable by specifying sysplex=on in your IOEFSPRM and then perform a rolling IPL)

Typically, if you enable zFS sysplex-aware for read/write zFS file systems, you see a performance improvement in most shared file system environments. There are, however, some servers that are not transparent to zFS sysplex-aware for read/write support. These are:

- The z/OS Distributed File Service SMB server cannot export zFS read/write file systems when zFS is running sysplex-aware on the same system the SMB server is running or on the system that owns the zFS file system.
- The Fast Response Cache Accelerator support of the IBM HTTP Server for z/OS V5.3 uses an API called register file interest (BPX1IOC 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.

If you are using either of these servers, you should remain running zFS non-sysplex aware for read/write (the default) for all sysplex members.

Restriction: You can consider running some systems with zFS as sysplex-aware and some with zFS as non-sysplex aware, but **this will cause a reduction in performance**. For details, see [“Running mixed zFS sysplex-aware and zFS non-sysplex aware” on page 13](#). Alternatively, you can allow these servers to access the data in a zFS read-only mounted file system, or you can store the data in a read/write mounted HFS file system.

Finally, there are some modifications to the way file system ownership works when you run zFS sysplex-aware. 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 31](#). There are some additional virtual storage requirements in the zFS address space used by default when you run zFS sysplex-aware. See the client_cache_size and client_reply_storage options specified in the IOEFSZPRM for default sizes. Also, additional storage is required for each open file.
Running all systems with zFS non-sysplex aware
If you decide to run zFS non-sysplex aware, shared file system support works as in prior releases. Ensure that you do not specify sysplex=on in your zFS IOEFSPRM configuration options file.

Running all systems with zFS sysplex-aware
If you decide to run zFS sysplex-aware for read/write, roll this support through all your sysplex members (this assumes all your members are using z/OS V1R11). You cannot change from zFS non-sysplex aware to zFS sysplex-aware dynamically. After changing IOEFSPRM, you must perform an IPL or a restart of zFS. IPL is the recommended method because in many ways it is less disruptive than a restart of zFS, which can cause zFS file systems to become unmounted. To enable zFS sysplex-aware, specify sysplex=on in the IOEFSPRM configuration file. You can specify it in a shared IOEFSPRM configuration file and each system picks up the specification in a rolling IPL. The sysplex option is ignored by previous releases.

During the migration to zFS sysplex-aware, for a time, some systems run zFS sysplex-aware and some run zFS non-sysplex aware. In this case, the benefits of zFS sysplex-aware are not fully realized. In fact, you will see a reduction in performance over the zFS non-sysplex aware configuration. See the topic about “Running mixed zFS sysplex-aware and zFS non-sysplex aware.” For example, a zFS read/write file system that is owned by one of the zFS sysplex-aware systems, but is accessed by one of the zFS non-sysplex aware systems does not see a performance benefit (for an example, see Figure 13 on page 37 and for the opposite see Figure 15 on page 38). The zFS read/write file systems that are owned and accessed by any of the systems that are running zFS sysplex-aware do typically see a performance benefit. After all your systems are running zFS sysplex-aware, you typically run as shown in Figure 8 on page 34.

Running mixed zFS sysplex-aware and zFS non-sysplex aware
You can decide to run with some systems running zFS sysplex-aware and some systems running zFS non-sysplex aware, but remember that this causes a reduction in performance. If you want to continue to run a prior release in your sysplex, you should remain running zFS non-sysplex aware. If you want to run the SMB server or the IBM HTTP Server with Fast Response Cache Accelerator you should remain running zFS non-sysplex aware. If you still want to run with some systems running zFS sysplex-aware and some systems running zFS non-sysplex aware, you will see a reduction in performance and you will have a more complex environment. You will need to be more concerned with z/OS UNIX ownership of zFS file systems. There are additional restrictions on file system ownership movement when some systems are running zFS sysplex-aware and some systems are running zFS non-sysplex aware. Typically, after a zFS file system is owned by a system running zFS sysplex-aware, the file system cannot be moved to a system running zFS non-sysplex aware without unmounting it, and then mounting the file system on the system running zFS non-sysplex aware.

In addition, there are performance implications with running in a mixed environment. In some cases, your performance might be worse than running all zFS file systems non-sysplex aware. One reason for this is because z/OS UNIX only performs caching of directory information on z/OS UNIX clients when the read-write file system is z/OS UNIX owned on a zFS non-sysplex aware system. If the read-write file system is z/OS UNIX owned on a zFS sysplex-aware system, z/OS UNIX does not perform directory caching and any z/OS UNIX client lookup request goes to the z/OS UNIX owning system through XCF communications rather than potentially being satisfied in the local z/OS UNIX directory cache. That means, explicitly choosing which system should be the z/OS UNIX owner of a read-write file system in a mixed environment is a trade-off between the following two options:

- Getting the benefit of zFS sysplex-aware among the zFS sysplex-aware systems by making it owned on a zFS sysplex-aware system.
- Getting the benefit of z/OS UNIX directory caching for all systems and no benefit of zFS sysplex-aware by making it owned on a zFS non-sysplex aware system.
As mentioned earlier, this is further complicated by the fact that after a zFS file system is z/OS UNIX owned on a system running zFS sysplex-aware, you cannot explicitly move it to a system running zFS non-sysplex aware. In general, if there are more systems running zFS sysplex-aware than not, it is most likely better to have the zFS read-write file system owned on a zFS sysplex-aware system for overall sysplex z/OS UNIX application performance. If you let z/OS UNIX choose the system to own the zFS read-write file system (for example, during shutdown or dead system recovery), z/OS UNIX chooses the z/OS UNIX owner randomly (unless, of course, you have specified a system list on MOUNT). Normally, after a read-write file system is z/OS UNIX owned on a system running zFS sysplex-aware, it is only moved to systems running zFS sysplex-aware.

---

**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, GI10-0670] or the 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 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.SIOELMOD) 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 IOEFSRPM member data set. 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 IOEFSRPM that has a single line that begins with an asterisk (*) in column 1. Subsequent modifications can be made to the IOEFSRPM member, see "IOEFSRPM on page 166".
      As an alternative to the IOEZPRM DDNAME specification, specify the IOEFSRPM member as a true PARMLIB member. In this case, the member has the name IOEPRMxx, where you specify xx in the parmlib member list. See "IOEFSRPM on page 166" for additional information about IOEPRMxx.
      To run zFS so that it is not under control of JES, see step 2 on page 15. 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))
      ADDUSER DFS OMVS(HOME('/opt/dfslocal/home/dfscntl') UID(0)) DFLTGRP(DFSGRP) AUTHORITY(USE) UACC(NONE)
      RDEFINE STARTED DFS.** STDATA(USER(DFS))
      RDEFINE STARTED ZFS.** STDATA(USER(DFS))
      SETROPTS RACLIST(STARTED)
      SETROPTS RACLIST(STARTED) REFRESH
      ```

**Note:** The DFS user ID must have at least ALTER authority to all VSAM LDS 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 LDS that contain zFS aggregates, you can assign the
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')
   ```

3. Run the `dfs_cpfiles` program.

   Running this program as described in the program directory is recommended even if you plan to only use the zFS support. The only zFS configuration file is the IOEFSPRM data set, 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 needed by the SMB or DCE DFS support. This avoids problems if the other support (SMB or DCE DFS) supplied by Distributed File Service is subsequently 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. Create or update the zFS parameter data set (IOEFSPRM).

   The zFS parameter data set is optional. The IOEZPRM DD can be omitted from the ZFS PROC or the IOEFSPRM data set can exist, with no parameters contained in it. Parameters are only required if you want to override the defaults for the zFS parameters. As mentioned previously, it is recommended that you create an empty IOEFSPRM member in a PDS. The IOEFSPRM member should have a single line in it that is a comment (an asterisk(*) in column 1). Update the IOEZPRM DD statement in the ZFS PROC to contain the name of the IOEFSPRM member. For example:

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

   If you are running in a sysplex, you must have different IOEFSPRM data sets for different systems.

   Chapter 5, "Using zFS in a shared file system environment," on page 31 explains the reasons to use different IOEFSPRM data sets. In this case, you should also specify a system qualifier in the data set name in the IOEZPRM DD. For example:

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

   As the 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. See "IOEFSPRM" on page 166 for additional information about IOEPRMxx.

   The PDS (organization PO) 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. Chapter 13, "zFS data sets," on page 165 contains a full description of the options that can be specified in IOEFSPRM.

5. Pre-allocate data sets for debugging - 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 19 for details on creating zFS file systems.

7. Create a directory and mount the zFS file system on it.
   A directory can be created with the z/OS UNIX `mkdir` command. (You can also 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 19 for details on mounting zFS file systems.

   **Note:** Steps 6 on page 15 and 7 can be repeated as many times as necessary for each permanently mounted zFS file system. Only step 6 on page 15 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.PR.V.COMPAT.AGGR001') TYPE(ZFS) MOUNTPOINT('/etc/mountpt')
   ```
Chapter 3. Managing zFS processes

This section describes the zFS address space and then discusses starting zFS, 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, you should 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. If you want to shut down an LPAR or you want to re-IPL an LPAR, use the MODIFY OMVS,SHUTDOWN operator command to shut down z/OS UNIX. This synchronizes data to the file systems and unmounts or moves ownership in a shared file system environment. A planned system shut down 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 F OMVS,SHUTDOWN, see the topic on Planned shutdowns using F OMVS,SHUTDOWN in z/OS UNIX System Services Planning, GA22-7800.

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, GA22-7800 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 shut down 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 (and all the zFS file systems) back to a consistent state. In addition, some file activity can be lost.

To restart zFS, reply r to message nn. (For example, r 1,r). 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 31 for information about shared file systems.

If the zFS colony address space has an internal failure, it will usually not terminate. It might disable an aggregate (see “Diagnosing disabled aggregates” on page 84). 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) will be issued and a reply will be requested.
If zFS 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, GA22-7800] 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.

To determine the file system owner, see

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 BPX0068I returns the PFS in one of the following possible states:

- A - the PFS is active. The timestamp is the start time of the PFS.
- I - the PFS is 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 - the PFS has stopped. It is waiting for a reply of R to restart or I to terminate the PFS.
- U - the PFS is unavailable.

For the complete message text, see [z/OS MVS System Messages, Vol 3 (ASB-BPX), SA22-7633]
Chapter 4. Creating and managing zFS file systems using compatibility mode aggregates

This section discusses creating compatibility mode aggregates and file systems. See Chapter 8, “Multi-file system aggregates,” on page 49 for information about multi-file system aggregates.

Note: Multi-file system aggregate support is not planned to be enhanced and might be removed sometime in the future.

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. This is a two step process:

1. Create a VSAM Linear Data Set using IDCAMS. 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 21 for additional information.

2. Format the VSAM LDS as a compatibility mode aggregate and create a file system in the aggregate using IOEAGFMT (see “ioeagfmt” on page 102 for additional information). When using ioeagfmt, it is required that the user must have ALTER authority to the VSAM LDS or must be UID 0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the z/OS UNIXPRIV class.

Note: You can also create a compatibility mode aggregate using the ISHELL or the automount facility. For more information on ISHELL, see z/OS UNIX System Services User’s Guide, SA22-7801. For more information on automount, see z/OS UNIX System Services Planning, GA22-7800.

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

The Control Interval (CI) size of a VSAM LDS that will be formatted as a zFS aggregate must be 4K. This is the default for IDCAMS and is, therefore, unspecified in the following example.

Figure 2 shows an example of a job that creates a compatibility mode file system.

```
//USERIDA JOB ,'Compatibility Mode',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//SYSDUMP DD SYSOUT=H
//AMSDUMP DD SYSOUT=H
//DASD0 DD DISP=OLD,UNIT=3390,VOL=SER=PRV000
//SYSIN DD *

/*
//CREATE EXEC PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate OMVS.PRV.COMPAT.AGGR001 -compat')
//SYSPRINT DD SYSOUT=H
//STDDOUT DD SYSOUT=H
//STDBERR DD SYSOUT=H
//SYSDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*/

Figure 2. Job to create a compatibility mode file system
```
Note, the `-compat` parameter in the CREATE step. That is what tells `IOEAGFMT` to create a compatibility mode file system. The result of this job is a VSAM LDS 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 LDS). The size of the zFS file system (that is, its quota) is based on the size of the aggregate.

The default for the size of the aggregate is the number of 8K blocks that fits in the primary allocation. You can specify a `-size` option giving the number of 8K 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 8K blocks that fits into the primary allocation, the VSAM LDS 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 the topic on “Growing a compatibility mode aggregate” on page 21. 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 LDS. 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 22. DFSMS decides when to allocate on these volumes during extension. Any VSAM LDS greater then 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, SC26-7410 for information about VSAM data sets greater than 4 GB in size. 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 LDS, it will only mount 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.

See Chapter 12, “zFS commands,” on page 101 for more information about `IOEAGFMT`.

Now, you can mount the zFS file system into the z/OS UNIX hierarchy with the TSO/E `mount` command. Here is an example of mounting the compatibility mode file system that was just created:

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

This assumes that the directory `/usr/mountpt1` exists and is available to become a mountpoint. See z/OS UNIX System Services Planning, GA22-7800 for complete information about mount points.

Here is an example of mounting the compatibility mode file system that was just created using the z/OS UNIX `mount` command:

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

When a zFS compatibility mode aggregate created prior to z/OS V1R7 is first mounted R/W on a higher release, the on-disk format is modified from a version 1.3 aggregate to a version 1.4 aggregate. This allows the performance of mount to improve (especially for zFS file systems with many files and directories).

During the automatic conversion, you will see messages such as:

```
IOEZ00500I Converting PLEX.JMS.AGGR007.LDS0007 for fast mount processing
IOEZ00518I Converting filesystem PLEX.JMS.AGGR007.LDS0007 to allow for fast mount
```
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 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 8K blocks and a (potential) secondary allocation of one cylinder. 180 8K blocks is 1440K bytes. A `zfsadm aggrinfo` command for this aggregate might show 1440K. This is a total of 1440K. `zfsadm grow` does this by calling DFSMS to allocate the additional DASD space. DFSMS requires a number of reserved blocks, therefore specify a few blocks larger than the current size before an allocation occurs. See the following example:

```
zfsadm aggrinfo omvs.prv.aggr003.lds0003
OMVS.PRV.AGGR003.LDS0003 (R/W COMP): 1286 K free out of total 1440
zfsadm grow omvs.prv.aggr003.lds0003 1440
IOEZ00173I Aggregate OMVS.PRV.AGGR003.LDS0003 successfully grown
OMVS.PRV.AGGR003.LDS0003 (R/W COMP): 1286 K free out of total 1440
```

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 1441
IOEZ00173I Aggregate OMVS.PRV.AGGR003.LDS0003 successfully grown
OMVS.PRV.AGGR003.LDS0003 (R/W COMP): 2006 K free out of total 2160
```

The aggregate now has a total size of 2160K bytes. You can specify 0 for the size to get a secondary allocation size extension. The file system quota has also been increased based on the new aggregate size. Aggregates cannot be made smaller.

Dynamically growing a compatibility mode aggregate

An administrator can specify that an aggregate should be dynamically grown if it becomes full. This is specified by the AGGRGROW PARM on the MOUNT command or globally by the aggrgrow option of the IOEFSPRM file (see [IOEFSPRM on page 166](#) for additional information). 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 allowed is based on DFSMS VSAM rules (see [DFSMS Using Data Sets, SC26-7410](#)). 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.

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.

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.

Chapter 4. Creating and managing zFS file systems using compatibility mode aggregates
Creating a multi-volume compatibility mode aggregate

To create a large zFS aggregate (for example, ten full volumes), you need all of the following:

- 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 (with 3336 cylinders free), that there are 15 tracks per cylinder and that you can get 6 8K blocks per track (15 x 6 = 90 8K blocks per cylinder), you should get 90 x 3336 = 300240 8K blocks per volume and 10 x 300240 = 3002400 8K blocks in the aggregate. Figure 3 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 (300240 8K blocks) multiple times until it reaches the total -size amount (3002400 8K blocks).

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. The following example job adds two volumes to the (SMS-managed) OMVS.ZFS.AGGR1 zFS aggregate:

```
//SUIMGVMA JOB (ACCTNO), 'SYSPROG', CLASS=A,
// MSGCLASS=X, MSGLEVEL=(1,1), NOTIFY=&SYSUID
//STEP01 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *

ALTER OMVS.ZFS.AGGR1.DATA -
   ADDVOLUMES(* *)
/*
   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 for Catalogs, SC26-7394] for additional 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.
```

Figure 3. Job to create a multi-volume compatibility mode aggregate
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). If a backup file system (created by the clone operation) is mounted, it must also be unmounted when the read-write file system is unmounted. Otherwise, the aggregate will not be detached. If only the read-write file system is mounted, you can use the remount capability of z/OS UNIX. For details, see the topic Remounting a mounted file system in z/OS UNIX System Services Planning, GA22-7800.

**Renaming or deleting a compatibility mode aggregate**

To rename a compatibility mode aggregate, use the IDCAMS ALTER command with the NEWNAME parameter. The aggregate must not be mounted to rename it.

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 following example 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.

```
//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.*)
```

To delete a compatibility mode aggregate, use the IDCAMS utility DELETE command. The aggregate must not be mounted to delete it. The following example deletes both the cluster name and the data component.

```
//SUIMGVMMD 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
```

See z/OS DFSMS Access Method Services for Catalogs, SC26-7394 for information and restrictions on IDCAMS ALTER NEWNAME and DELETE.

**Unmounting zFS file systems before mounting, copying or backing up**

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 will wait 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 the another system. If the time stamp is not updated, the mount succeeds after waiting for 65 seconds.

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. In the case that the aggregate is being mounted on a different member in the same sysplex after a failure, zFS will 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.

Cloning a file system

zFS provides the ability to clone a file system by using the `zfsadm clone` command. When you clone a file system, you create a copy of the file system in the same aggregate. There must be enough physical space available in the aggregate for the clone to be successful. This copy of the file system is read-only and is called a backup file system. The name of the backup file system is the same as the original (read-write) file system with `.bak` (in lower case) appended to the file system name. This means that you need to limit the length of the file system name to 40 characters if you want to clone it.

Here is an example of a `zfsadm clone` command:

```
zfsadm clone -filesystem OMVS.PRV.FS1
```

IOEZ00225I File system OMVS.PRV.FS1 successfully cloned.

Initially, a backup file system takes up a relatively small amount of space because only the metadata is copied, not the user data. The backup file system’s data block pointers point to the same data blocks that the read-write file system’s data block pointers point to. After a clone operation, if the read-write file system user data is updated, zFS ensures that new physical blocks are allocated to hold the updates, while maintaining the backup file system’s data pointers to the original data. The backup file system remains a point-in-time read-only copy in the face of updates to the read-write file system. This backup file system can be mounted (read-only) so that users with the authority to perform mounts of the read-write file system can have an online backup of that file system available. That is, if a user accidently erases a file from the read-write file system, they can simply copy the file from the backup into the read-write file system to restore the file to the time the backup was created.

Here is an example of a TSO/E MOUNT command for the backup file system:

```
MOUNT FILESYSTEM('''OMVS.PRV.FS1.bak''') MOUNTPOINT('/etc/mountpt3') TYPE(ZFS) MODE(READ) NOAUTOMOVE
```

Here is an example of the z/OS UNIX `mount` command for the backup file system:

```
/usr/sbin/mount -t ZFS -r -a no -f OMVS.PRV.FS1.bak /etc/mountpt3
```

The read-write file system can be cloned again (re-clone). When a backup file system already exists, it is replaced during the clone operation. One backup file system can exist for each read-write file system. Backup file systems cannot be mounted during the clone operation.

If a failure occurs during a clone of the file system (causing the clone to be only partially created), zFS will delete the partial clone on the next mount. If the file system contains many files, the delete of the partial clone might delay the mount.

You can clone or re-clone a set of file systems with the `zfsadm clonesys` command. This can be specified in terms of file systems with a file system name prefix or file systems in an aggregate or both.
zFS disk space allocation

A zFS aggregate is an array of 8K blocks. There following 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** - This file records metadata changes. Its size is 1% of the disk size by default.
- **Bitmap** - This file lists the blocks that are free on disk. The file size is dependent on the size of the aggregate.
- **Aggregate File System List** - This list describes the file systems contained in the aggregate. For compatibility mode aggregates it is usually only one 8 KB block. For multi-file system aggregates, its size depends on how many file systems are in the aggregate.

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 1KB blocks. If you use the `-long` option of the `zfsadm aggrinfo` command, it shows the number of free 8K 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.

zFS aggregates are all capable of containing multiple file systems, even compatibility mode aggregates. Compatibility mode aggregates can have backup file systems in them that take space if the clone operation is used. Each file system has a quota represented in 1KB fragments. The quota of a file system is a logical number and can be smaller or larger than the size of the disk (if the size of the disk were expressed in 1KB fragments).

The file system quota for compatibility mode aggregates is set to be the total aggregate size (in 1 KB units).

The `zfsadm lsquota` command will show the quota in 1KB units and will also show the aggregate size and usage in 1 KB units (it shows the amount of space used for the three special objects above also).

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

zFS stores files on disk in one of three ways:

- **inline** - if the file is 52 bytes or less, its 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** - if the file is 7 KB or less and has never been larger than 7 KB, it is stored in 1 KB fragments (hence it is stored in part of an 8 KB block). Multiple small files can share the same 8 KB block on disk.
- **blocked** - if the file is over 7 KB, it is stored as an array of 8 KB blocks.

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. As shown in Figure 4 on page 26 for example, if there were two 8 KB blocks and twenty 1 KB blocks left, `df -k` reports 36 KB available.
Because it is a combination of 8 KB blocks and 1 KB blocks, it can happen that there are many 1 KB blocks available but no 8 KB blocks left. As shown in Figure 5 for example, if there were 0 8KB blocks left and 20 1KB blocks available, `df -k` reports 20 KB available. Now, 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.

There are other rules that 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 1KB blocks available, the file system will report that there is no space for the file. Also, a file stored as fragments cannot share the same 8KB 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 8KB blocks. The `aggrfull` option of MOUNT, zfsadm attach, IOEFSPRM and `define_aggr` indicates the amount of free 8KB blocks. If you are out of 8KB 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 will happen 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 LDS). 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. However, if you have cloned the file system, you will not be able to free space by erasing files. A cloned file system requires additional space in the aggregate to erase a file. In this case, you might be able to free some space by recloning the file system.

Note that because of the difference between how HFS and zFS manages 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 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 31 and review “Unmounting zFS file systems before mounting, copying or backing up” on page 23.

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.

There is limited support for a zFS file system to be mounted read-write to one system and mounted read-only on another system.

- **Example:** If you mount a zFS file system read-write to a system (system A), you cannot mount that file system read-write to any other system (system B). If you attempt to mount the file system read-write to another system (system B), the mount will fail.

- **Example:** If you mount a zFS file system read-write to a system (system A), and then you subsequently mount the file system read-only on another system (system B), the mount will succeed if no data has been written to the file system because it was attached read-write (to system A). If data has been written, you will receive reason code EFxx6271 indicating that the log must be replayed and the read-only mount (to system B) will fail.

  If you specify `romount_recovery=on`, the mount will fail with reason code EF096058 because the mount was an attempt to (temporarily) mount the file system read-write.

However, the examples shown here only work when:

- You do not share Global Resource Serialization (GRS) between these systems. (When you share GRS across the systems, the first mounted read-write data set is allocated as EXCL, and the second mounted read-only data set tries to allocate as SHR.)
- You share the catalog or you have a way to get the zFS file system data set name into the other catalog.

Beginning with z/OS V1R11, these examples work even when sharing GRS between the systems because zFS allocates the aggregate as SHR.

Consider the following limitations of this support.

**Notes:**

1. If you are running z/OS V1R5 or higher on the system where the file system is mounted read-write, you can use the remount capability to clear the log. See “Disabled compatibility mode aggregate” on page 84.

2. If you are running z/OS V1R6 or higher on the system where the file system is mounted read-write, you can quiesce and unquiesce the aggregate and clear the log. For example,

   ```
   zfsadm quiesce -aggrname name
   ```

   and then
Otherwise, you need to unmount the read-write file system from the system (system A) and then mount the file system read-write on that same system (system A). This will clear the log and permit it to be mounted read-only. After the file system is successfully mounted read-only, you will receive a message (informational message “IOEZ00439I Read-only aggregate aggrname is attached read-write on another system”) and errors can occur on the read-only file system (system B) if writes have occurred on the file system from the system where it is read-write mounted (system A). To recover from the errors on the read-only file system, you need to unmount the read-write file system (system A) and then remount it read-write (system A). Then you can unmount the read-only file system and then remount it read-only (system B).

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 × 6 tracks). This only leaves 143 KB of free space available for files and directories (see the example below). 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 “zFS disk space allocation” on page 25). IBM does not recommend using such small file systems. You can permit the file system to grow automatically (you must have aggrgrow=on in IOEFSPRM file or in the MOUNT PARM and you must have a secondary allocation specified on the define - specified as 5 in the example below). However, your log file size is very small and might cause contention. The log file size cannot be increased after the aggregate is formatted.

```
# zfsadm define -aggr PLEX.JMS.AGGR007.LDS0007 -volumes CFC000 -tracks 6 5
IOEZ00248I VSAM linear dataset PLEX.JMS.AGGR007.LDS0007 successfully created.

# zfsadm format -aggr PLEX.JMS.AGGR007.LDS0007 -compat
IOEZ00077I HFS-compatibility aggregate PLEX.JMS.AGGR007.LDS0007 has been successfully created
# /usr/sbin/mount -t ZFS -f PLEX.JMS.AGGR007.LDS0007 -o 'AGGRGROW' /zfsmnt3
# zfsadm aggrinfo -aggr PLEX.JMS.AGGR007.LDS0007 -long
PLEX.JMS.AGGR007.LDS0007 (R/W COMP): 143 K free out of total 288 version 1.4
    17 free 8k blocks;    7 free 1K fragments
    112 K log file;       8 K filesystem table
    8 K bitmap file
```

The architected maximum zFS compatibility mode aggregate size is approximately 4 TB (1KB × 4GB). If you use 3390s that have 65520 cylinders per volume, you can create a compatibility mode aggregate of about 2,850,088,550,400 bytes: 65520 cylinders per volume × 90 blocks per cylinder × 8KB per block × 59 volumes = 2654 GB = 2.59 TB. The usable free space in the file system would be a small amount less than this. However, if you plan to do this, you should consider the implications of backup and recovery for failures (media failure, data corruption, and others).

**Restriction:** A zFS compatibility mode aggregate is limited to 4 TB even on Extended Address Volume (EAV) devices. The maximum number of objects (files, directories and ACLs) in a zFS file system is 4 billion. The maximum size of a file is a little less than the size of the file system. The maximum size of a directory is 4 GB. There is a limit of 65533 (64K-3) subdirectories in a 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 directory. For best performance, attempt to spread names among many directories.

If you have long response times, you can get a first indication whether it might involve a directory size problem by examining the output of the `MODIFY ZFS,QUERY,KN` operator command or the z/OS UNIX `zfsadm query -knfps` 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 \texttt{largedir.pl} command to help determine which directories are
large. It will report any zFS directories that have a large size. Find the \texttt{largedir.pl} command on the z/OS
UNIX Tools and Toys Web page at \url{http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1toy.html} To
determine how large a particular directory is (how many bytes the directory contains), use the \texttt{ls -ld}
command against the directory to display its size in bytes.

For example, if you suspect \texttt{/zfsmnt5/testdir} is too large, issue a command similar to the following:

\begin{verbatim}
# ls -ld /zfsmnt5/testdir
drwxr-xr-x 2 G0DOUG AUDIT 1638400 Jan 18 2007 /zfsmnt5/testdir
\end{verbatim}

The output shows \texttt{/zfsmnt5/testdir} is over 1 MB in size and contains many names (or at one time
contained many names).

After a zFS directory becomes too large, the space is not reclaimed when names are removed from the
directory. Therefore, you must look at the size of the directory rather than the number of names it currently
contains. The only way to reclaim the space is to remove the directory itself rather than erasing names
within it. So if the directory currently has a small number of names, but is large in size, try using the
following set of commands to make a new directory:

\begin{verbatim}
mkdir /zfsmnt5/testdir2
cp /zfsmnt5/testdir/* /zfsmnt5/testdir2
-or-
/samples/copytree /zfsmnt5/testdir /zfsmnt5/testdir2 (if testdir has subdirectories)
rmdir /zfsmnt5/testdir
mv /zfsmnt5/testdir2 /zfsmnt5/testdir
\end{verbatim}

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 must copy the file
system to another (new) file system and delete the original file system. See Chapter 7, “Migrating data
from HFS to zFS,” on page 47 for information about copying one file system to another.

When you must have a large number of file names in a single directory, it is best to use an HFS file
system directory for that application.
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 and all zFS file systems must be compatibility mode file systems (that is, they cannot be file systems in multi-file system aggregates). zFS multi-file system aggregates are not supported by automount.

zFS sysplex-aware support has no effect on z/OS UNIX file system movement definitions. z/OS UNIX still performs movements in the sysplex-aware group of systems independent of zFS ownership movement.

**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

Beginning with z/OS V1R11, you can enable zFS to run sysplex-aware for read/write mounted file systems. zFS (and HFS) already run sysplex-aware for read-only mounted file systems. Your systems must be running zFS V1R11 and above to support a zFS sysplex-aware for read/write mounted file systems environment. For details, see *Two-step APAR procedure for z/OS V1R11* on page 11.

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"
- "zFS support for read/write non-syplex aware mounted file systems" on page 32
- "zFS support for read/write sysplex-aware mounted file systems" on page 33
- "zFS ownership versus z/OS UNIX ownership of file systems" on page 34
  - "Determining the file system owner" on page 35
- "When is the z/OS UNIX owner important?" on page 36
- "What happens when one system runs zFS non-syplex aware?" on page 36
- "What happens when a zFS non-syplex aware system owns a file system and the other systems are running sysplex-aware?" on page 37
- "Dynamic movement of the zFS owner" on page 38

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 (as seen in Figure 6). 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-syplex aware mounted file systems**

This section applies to releases before z/OS V1R11 or if you do not explicitly configure zFS to run sysplex-aware in z/OS V1R11.

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 path length 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

Beginning with z/OS V1R11, you can enable zFS to run sysplex-aware for zFS read/write file systems. For details about when this is appropriate, see "Enabling the zFS sysplex-aware function" on page 12. When a file system is mounted read/write (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, file requests are sent directly to zFS which in turn function ships the request to the zFS owning system. In many cases, zFS will not need to function ship the request to the owning zFS system because the data will be in the local cache. zFS caches data to avoid frequent forwarding to an owner, and this typically improves performance for most workloads. The zFS owning system is the only system that does I/O to the file system. In this case, zFS is running sysplex-aware for read/write mounted file systems.

**Note:** In z/OS V1R11, zFS continues to support read-only mounted file systems as shown in Figure 6 on page 32.
zFS ownership versus z/OS UNIX ownership of file systems

As a result of zFS running sysplex-aware for read/write mounted file systems, zFS takes responsibility for function shipping file requests. This means that zFS must have the indication of a file system owner to receive the function ship file requests. That system is the **zFS owner**, z/OS UNIX still has its indication of owner, which is the z/OS UNIX owner. The zFS owner is independent of the **z/OS UNIX owner**. The zFS owner is the system that does the actual file I/O. 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 36).

In Figure 8, 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 9 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.

Figure 8. zFS sysplex-aware for read/write mounted file systems

Figure 9. zFS sysplex-aware with new owner
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`
- `d omvs,f`
- `fbpoinit,filesys=d,all`.

The following example shows the output of the `zfsadm lsaggr` command and the `df -v` command after the file system was mounted (as shown in Figure 8 on page 34) issued from SY2:

```bash
# 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 4294917290 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 10. zfsadm lsaggr and df -v output after mount**

The following example shows the output of the `zfsadm lsaggr` command and the `df -v` command after the file system was moved (as shown in Figure 9 on page 34) by both z/OS UNIX and zFS and SY2 has come back up. The `zfsadm lsaggr` and `df -v` commands are issued from SY2:

```bash
# 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 11. zfsadm lsaggr and df -v output after movement**

Notice two important points:

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

This is because zFS is running sysplex-aware and file requests are not function shipped by z/OS UNIX. The file requests are function shipped by zFS to the zFS owner. Even though each local catch-up mount causes zFS to open the file system data set for read/write, only the zFS owner actually reads and writes to the file system. The file system is opened on each system in order to prepare for ownership takeover in case a system needs to do that.

**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, use `D GRS,RES=(SYSZIOEZ,IOEZLT.file_system_name)` to display the system name of the zFS owner as the exclusive owner of the resource name. For more information, see the serialization summary in **z/OS MVS Diagnosis: Reference, GA22-7588**
When is the z/OS UNIX owner important?

When a zFS 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 will treat that system (such as SY1) as a client and function ship 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 will indicate `Client=Y` for that file system. In turn, zFS function ships the request 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.

The zFS owner can be different than the z/OS UNIX owner. In this case, the request might be function shipped twice—one time by z/OS UNIX and one time by zFS—unless it is already in the cache. In many cases, zFS will not need to function ship the request to the owning zFS system because the data will be in the local cache. zFS caches data to avoid frequent forwarding to an owner, and this typically improves performance for most workloads.

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

![Figure 12. File system ownership when mount fails](image)

What happens when one system runs zFS non-sysplex aware?

When zFS run non-sysplex aware (as it does in releases before z/OS V1R11 and in z/OS V1R11 with the `sysplex=off` default) on a system (such as, SY1) and a zFS file system is mounted on a zFS sysplex-aware system (such as SY2), z/OS UNIX on SY1 will not send a local mount to zFS and the system will become a client to the z/OS UNIX owner (SY2), similar to Figure 13 on page 37.

You can determine whether zFS on a particular system is running sysplex-aware or not by using the `MODIFY ZFS,QUERY,LEVEL` operator command. If the command indicates `sysplex(file)`, zFS is sysplex-aware, or if the command indicates `sysplex(admin-only)` zFS is not sysplex-aware. For examples, see "Determining service levels" on page 78. You can also use the `zfsadm configquery -sysplex_state` command, which indicates 2 if zFS is running sysplex-aware or 1 if it is not. See "zfsadm configquery" on page 131.
The zFS owner can be different than the z/OS UNIX owner. In this case, the request might be function shipped twice—one time by z/OS UNIX and one time by zFS—unless it is already in the cache. In many cases, zFS will not need to function ship the request to the owning zFS system because the data will be in the local cache. zFS caches data to avoid frequent forwarding to an owner, and this typically improves performance for most workloads.

**Figure 13. One system non-sysplex aware**

The zFS owner can be different than the z/OS UNIX owner. In this case, the request might be function shipped twice—one time by z/OS UNIX and one time by zFS—unless it is already in the cache. In many cases, zFS will not need to function ship the request to the owning zFS system because the data will be in the local cache. zFS caches data to avoid frequent forwarding to an owner, and this typically improves performance for most workloads.

**Figure 14. Non-sysplex aware with different owners**

**What happens when a zFS non-sysplex aware system owns a file system and the other systems are running sysplex-aware?**

If a file system is mounted on a zFS system that is running non-sysplex aware (such as SY1), and the other systems are running zFS sysplex-aware, the file system is mounted successfully, but the local catch-up mounts on the other (sysplex-aware) systems will fail. The failure causes z/OS UNIX to treat those systems as clients and z/OS UNIX will function ship file requests to the z/OS UNIX owner (SY1). This is not apparent to the user except informational message BPXF221I is issued by z/OS UNIX on the sysplex-aware systems and access to that file system gets none of the performance benefit from running zFS running sysplex-aware. IBM Health Checker for z/OS also supplies the USS_CLIENT_MOUNTS check, which generates an exception message when it finds a file system that is function shipping, but can be mounted locally. See the topic on USS_CLIENT_MOUNTS in [IBM Health Checker for z/OS: User's Guide, SA22-7994](https://www.ibm.com).
In the case of a sysplex-wide IPL, if you have a mixture of sysplex-aware zFS systems and non-sysplex aware systems in your shared file system environment, the order in which you IPL the systems can affect whether z/OS UNIX ownership of read/write file systems is on one of the sysplex-aware systems or on one of the non-sysplex aware systems. Typically, it is desirable to have z/OS UNIX ownership (and therefore zFS ownership) be on one of the sysplex-aware systems. This allows zFS to use its sysplex-aware support to attempt to improve performance. If you IPL a non-sysplex aware system first (SY1), read/write file systems are typically owned by that non-sysplex aware system (SY1) and subsequent systems that IPL (sysplex-aware systems SY2 and SY3) are required to function ship requests to the z/OS UNIX owning system (SY1). See Figure 15.

If this situation occurs, you can move the z/OS UNIX ownership of the file system onto one of the sysplex-aware systems by using the SETOMVS command or the chmount z/OS UNIX command. For example, if file system PLEX.JMS.AGGR006.LDS0006 is mounted at /zfsmnt5, you can issue SETOMVS FILESYS,FILESYSTEM='PLEX.JMS.AGGR006.LDS0006',SYSNAME=SY2 or /usr/sbin/chmount -d SY2 /zfsmnt5 to move z/OS UNIX ownership to SY2. For information about how to determine the z/OS UNIX owner and the zFS owner of a file system, see “Determining the file system owner” on page 35.

Using the SMB server

There might be cases, however, when you need to have a file system be z/OS UNIX owned by a non-sysplex aware system. The z/OS SMB server can export zFS file systems, but they cannot be z/OS UNIX owned by a system running zFS sysplex-aware. For example, if a file system is to be exported by the z/OS SMB server, it must be z/OS UNIX owned by a non-sysplex aware system. Of course, system specific file systems must be z/OS UNIX owned by that system.

Dynamic movement of the zFS owner

When zFS runs sysplex-aware on multiple (or all of the) systems in the sysplex, an important aspect of performance is which system is the zFS owner. The zFS owner is the system that does all I/O to the file system. zFS automatically moves the zFS owner among zFS sysplex-aware systems based on the number of I/O requests from each system.

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:

Source system
22.19.12 DCEIMGVM IOEZ00548I Requesting that DCEIMGVM takeover aggregate PLEX.JMS.AGGR006.LDS0006

Figure 15. zFS non-sysplex aware system with sysplex-aware systems
For zFS sysplex-aware 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 systems that are running zFS sysplex-aware and 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. IBM recommends mounting zFS file systems with UNMOUNT instead of NOAUTOMOVE.

**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. IBM recommends mounting 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 does the actual I/O.

If zFS is running non-sysplex aware (sysplex=off), 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 mountpoints have different results in such cases because the zFS file system is not mounted on those systems not running zFS.
- zFS file systems 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.
- Although the clone operation is allowed in a compatibility mode aggregate, if both file systems (the read/write and the backup file systems) are mounted, some restrictions and limitations apply to the compatibility mode aggregate because there are really two mounted file systems in the aggregate. `chmount` and `setomvs` cannot be used to move ownership. `MOUNT` (that is, the mount of the second file system from a different system) and `AUTOMOVE` (if you do not have incompatible `AUTOMOVE` SYSTEM LISTs) work in z/OS V1R4 and above.
- In order to share IOEFSPRM across a sysplex, the following specifications must use system symbols to differentiate the data set names:
  - trace_dsn
  - msg_output_dsn

Chapter 5. Using zFS in a shared file system environment
– multi-file system aggregate

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 only using compatibility mode aggregates (and file systems), and 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. You can also use system symbols in the define_aggr option of IOEFSPRM.

As an 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 a 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 166 for more information about IOEFSPRM.

The following information describes z/OS UNIX considerations that relate to the level of z/OS running on the members of the sysplex:

• When all members of the sysplex are at z/OS V1R2 or later and some or all systems are running zFS:
  – All systems running zFS see zFS file systems. The file system hierarchy appears differently when viewed from systems with zFS mounted file systems than it does from those systems not running zFS. The path name traversal through zFS mountpoints have different results in such cases because the zFS file system is not mounted on those systems 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 file systems that are noautomove, become unowned.
    - 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 file system that is mounted and unowned, the file system must be unmounted.
    - The 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 fails to find another z/OS UNIX owner, the file system and all file systems mounted under it are unmounted in the sysplex.
    - zFS file systems that are noautomove and all file systems mounted under them are unmounted in the sysplex.

• In z/OS V1R7 and above, zfsadm commands work across the shared file system environment. You can display and modify zFS aggregates and file systems using zfsadm from any member of the sysplex regardless of which member owns the aggregate.

Multi-file system aggregates and shared file systems

Beginning with z/OS V1R11, you cannot attach a zFS multi-file system aggregate in a shared file system environment. You can attach a compatibility mode aggregate (in all environments). An attached compatibility mode aggregate will still show up as MULT if you issue zfsadm aggrinfo against it because it is attached and not mounted. However, beginning with z/OS V1R11, in a shared file system environment you cannot perform any tasks that make attached compatibility mode aggregates become a multi-file
system aggregates (that is, you cannot use `zfsadm create` to create another file system, you cannot use
`zfsadm delete` to delete the read/write file system, you cannot use `zfsadm rename` to rename the
read/write file system, and so on). Message IOEZ00552I is not issued when a compatibility mode
aggregate is attached in z/OS V1R11 and above in a shared file system environment because zFS
recognizes that it is an attached compatibility mode aggregate.

In a shared file system environment that contains releases earlier than z/OS V1R11, there can be attached
multi-file system aggregates that are owned by a system running an earlier release.

In releases before V1R11, multi-file system aggregates have very limited support in a shared file system
environment. Multi-file system aggregates can be attached, but file systems that are in multi-file system
aggregates cannot be mounted. You should use compatibility mode aggregates only. If you have data in
file systems that are in multi-file system aggregates, you should copy each file system into a compatibility
mode aggregate using a prior release of z/OS or a non-shared file system environment. See Chapter 7,
"Migrating data from HFS to zFS," on page 47 for more information about copying data from one file
system to another.
Chapter 6. Performing a back up of zFS

This section describes how to back up a zFS aggregate using a DFSMSdss™ logical dump. DFSMSdss automatically performs a quiesce of the zFS aggregate before dumping the data set and an unquiesce when the dump ends. File systems in the aggregate can be mounted or not mounted. Before performing a back up, review the information in "Unmounting zFS file systems before mounting, copying or backing up" on page 23 and the following guidelines.

Guidelines: Review the following guidelines before performing a back up of zFS:

1. Full volume dumps of mounted file systems do not quiesce the user. 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, SC26-7402.

2. 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).

3. If any systems in the sysplex are running a release before z/OS V1R7, the job must be run on the sysplex member where the aggregate or aggregates are attached. If the job is not run on the same member of the sysplex, the quiesce fails and the job stops. However, if all systems in the sysplex are running z/OS V1R7 or later, and SYSPLEX(YES) is specified in the BPXPRMxx member, indicating the system is participating in z/OS UNIX shared file system, you can run the backup job on any member of the sysplex.

4. If a quiesce is not done before the backup, 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 44, 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 16 on page 43.

```c
/*/-----------------------------------------------------------------
// THIS STEP QUIESCES THE AGGREGATE.
//*************************************************************************-
//QUIESCE EXEC PGM=IOEZADM,REGION=0M,
// PARM=('quiesce -aggregate hlq.ZFS.AGGR004')
//*
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR 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
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*

Figure 16. Steps for quiesce and unquiesce
```
Backing up a zFS aggregate

Figure 17 shows an example of a job for backing up a zFS aggregate (and all the file systems). Ensure 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, SC26-7402.

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.PRV.AGGR005.LDS0005). Figure 18 is an example of a job to restore a zFS aggregate.

Compatibility mode aggregate

After the aggregate is restored, perform the following steps:
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).

Multi-file system aggregate

After the aggregate is restored, perform the following steps:
1. Unmount the file systems in the original aggregate (if any are mounted).
2. Detach the original aggregate (in this case, hlq.ZFS.AGGR004) if it still exists.
3. Attach the restored aggregate (in this case, OMVS.PRV.AGGR005.LDS0005).
4. Mount the file systems in the restored aggregate.

Another example of a logical restore of a zFS aggregate using DFSMSdss by replacing the existing aggregate is shown. 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. Figure 19 is an example of a restore replace job.

```
//ZFSREST2 JOB (OS390),'PROGRAMMER',CLASS=A,
// MSGCLASS=X,MSGLEVEL=(1,1)
//****************************************************************************
//* THIS JOB RESTORES A ZFS AGGREGATE.
//****************************************************************************
//ZFSREST EXEC PGM=ADDRDSSU,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//INDS DD DISP=SHR,DSN=SUIMGUR.ZFS.DUMP1
//SYSSIN DD *
   RESTORE DATASET(INCLUDE(hlq.ZFS.AGGR004)) -
       CATALOG -
       REPLACE -
       WRITECHECK -
       INDD(INDS)
```

Figure 19. Job to restore a zFS aggregate with replace

For additional details on DFSMSdss logical restore, see z/OS DFSMSdss Storage Administration, SC35-0423
Chapter 7. Migrating data from HFS to zFS

This section discusses how to migrate data from HFS to zFS.

Guideline: Do not use the HFS to zFS migration tool if you are migrating your sysplex root. To migrate the sysplex root, consider using the MODIFY OMVS,NEWROOT operator command. For details, see the topic Steps for dynamically replacing the sysplex root file system in z/OS UNIX System Services Planning, GA22-7800.

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.

Note: The number of blocks to store a zFS file system might not be exactly the same as HFS.

For additional information about migrating data from HFS to zFS see z/OS Migration, GA22-7499.

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, SA22-7802 for more information about the pax command. When the data is being copied, the file system being accessed must be mounted. You can also use pax to copy a file system that resides in a multi-file system aggregate into a compatibility mode aggregate. You must do this using a prior release or a non-shared file system environment because file systems that are in multi-file system aggregates cannot be mounted when running in a shared file system environment.

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/dce/testzfs1. You issue the following commands from z/OS UNIX:

1. Move to the source (HFS) file system mounted at /etc/dfs
   ```sh
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
   ```sh
   pax -wvf /tmp/zfs1.pax
   ```
3. Move to the target (zFS) file system mounted at /etc/dce/testzfs1
   
   cd /etc/dce/testzfs1

4. Read the archive file into the zFS file system mounted at /etc/dce/testzfs1
   
   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/dce/testzfs1. You issue the following commands from OMVS:

1. Move to the source (HFS) file system mounted at /etc/dfs
   
   cd /etc/dfs

2. Copy the (HFS) file system mounted at /etc/dfs to the (zFS) file system mounted at /etc/dce/testzfs1
   
   pax -rwvCDM -p eW . /etc/dce/testzfs1
Chapter 8. Multi-file system aggregates

This section discusses multi-file system aggregates, however, it is recommended that you use compatibility mode aggregates because they are more like HFS file systems. See Chapter 1, “zSeries File System (zFS) overview,” on page 3. Compatibility mode aggregates have a single file system in the aggregate and are fully supported in a sysplex (shared file system) environment. See Chapter 4, “Creating and managing zFS file systems using compatibility mode aggregates,” on page 19 for information about zFS compatibility mode aggregates and file systems.

Restrictions:
- In z/OS V1R8 and above, zFS file systems contained in multi-file system aggregates cannot be mounted in a shared file system environment. Multi-file system aggregate support is not planned to be enhanced and will be removed sometime in the future.
- In z/OS V1R11 and above, you cannot attach a zFS multi-file system aggregate in a sysplex shared file system environment. Multi-file system aggregates are not supported in a sysplex shared file system environment. See Chapter 5, “Using zFS in a shared file system environment,” on page 31 for information about shared file systems.

With multi-file system aggregates, the administrator can create multiple file systems in a single aggregate. This allows space sharing between different file systems in the same aggregate. Therefore, if files are being deleted from one file system, another file system (in the same aggregate) can use that physical space for creating new files.

Creating a multi-file system aggregate

A multi-file system aggregate is a VSAM Linear Data Set (LDS) that can contain multiple zFS file systems. The multi-file system aggregate and the zFS file systems that are contained in the aggregate are created separately. First, the multi-file system aggregate is created using the zFS ioeagfmt utility. The aggregate must be attached and then one or more zFS file systems are created in the aggregate using one or more zfsadm create commands. Creating a zFS multi-file system aggregate is a two step process:

1. Create a VSAM LDS using IDCAMS. The VSAM Linear Data Set must have a secondary allocation size specified, if you want to use dynamic grow. See “Dynamically growing a multi-file system aggregate” on page 53 for additional information.
2. Format the VSAM LDS as a multi-file system aggregate using ioeagfmt.

The VSAM LDS and the zFS multi-file system aggregate both have the same name and that name is equal to the VSAM LDS cluster name.

Figure 20 on page 50 shows an example of a job that creates and formats a zFS multi-file system aggregate.
After the multi-file system aggregate is formatted, it contains zero zFS file systems in it. The size of the aggregate is reported by `ioeagfmt` as the number of 8K blocks that fit into the primary allocation or as specified on the `-size` option. The multi-file system aggregate must then be attached on a system before any `zfsadm` commands can be issued against it.

The default for the size of the aggregate is the number of 8K blocks that fits in the primary allocation. You can specify a `-size` option giving the number of 8K 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 8K blocks that will fit into the primary allocation, the VSAM LDS is extended to the size specified. (You might need to specify the `-grow` option if the extension will not fit on a single volume.) A secondary allocation on the VSAM LDS is not required. 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 LDS. DFSMS decides when to allocate on these volumes during extension. VSAM LDSes greater than 4 GB can be specified by using the extended format and extended addressability capability in the data class of the data set.

z/OS DFSMS Using Data Sets, SC26-7410 contains information about VSAM data sets greater than 4 GB in size. zFS does not support the use of a striped VSAM Linear Data Set as a zFS aggregate. If you attempt to attach a zFS aggregate that had previously been formatted and is a striped VSAM LDS, it will only attach as read-only.

When you attach a multi-file system aggregate on a system, the zFS Physical File System (PFS) must be active on that system. See Chapter 3, “Managing zFS processes,” on page 17 for information about starting zFS. You can attach using one of the following methods:

- The `zfsadm attach` command can be issued on that system, or
- A `define_aggr` statement can be placed in the IOEFS PRM file for that system and the zFS PFS can be started (or restarted).

After a new multi-file system aggregate is defined and formatted, the zFS administrator attaches it by issuing the `zfsadm attach` command and then adds a `define_aggr` statement for the aggregate in that system’s IOEFS PRM file so that the aggregate is automatically attached each time the zFS PFS is subsequently started (or restarted). The `define_aggr` statement does not need to be added to the IOEFS PRM file, but then the aggregate must be attached (by using the `zfsadm attach` command) each time the zFS PFS is started (or restarted).
The `zfsadm attach` command for the multi-file system aggregate just created is shown in the following example:
```
zfsadm attach -aggregate omvs.prv.multi.aggr002
```

A `define_aggr` statement in the IOEFSPRM file for the multi-file system aggregate just created is shown in the following example:
```
define_aggr cluster(omvs.prv.multi.aggr002)
```

**Note:** zFS aggregate names are not case sensitive. Because the names are always VSAM LDS names, they are always folded to upper case.

After the multi-file system aggregate is attached, the administrator can now create zFS file systems in the aggregate. This is accomplished using the OMVS `zfsadm create` command. The following example shows an example of creating a file system in the aggregate you just created and attached:
```
zfsadm create -filesystem OMVS.PRV.FS1 -aggregate omvs.prv.multi.aggr002 -size 5000
```

The previous example creates a zFS file system (named `OMVS.PRV.FS1`) in the OMVS.PRV.MULTI.AGGR002 aggregate. The file system has a maximum size of 5000 1K blocks.

**Note:** zFS file system names are case sensitive. The file system name specified on the `zfsadm create` command is not folded to upper case. If you create a zFS file system using lower case letters, it must be mounted using these same lower case letters. To do this, use the TSO/E MOUNT command and surround the file system name with a pair of three single quotation marks. See [z/OS UNIX System Services Command Reference, SA22-7802](https://www.ibm.com/support/knowledgcenter/SR295G_2.2.0/com.ibm.zos.v2r2.doc/adm/stdiohtml) for information about the TSO/E MOUNT command.

If you are using both multi-file system aggregates and compatibility mode aggregates, do not name any file systems in multi-file system aggregates with the same name as any of your compatibility mode aggregates. If you do this, you will get a different file system mounted depending on whether an aggregate is attached or not. For example, suppose you have compatibility mode aggregate A.B.C and you have multi-file system aggregate D.E.F that contains file system A.B.C and you have multi-file system aggregate D.E.F that contains file system A.B.C. When you mount file system A.B.C, you will get the one in aggregate D.E.F mounted if D.E.F is attached. If D.E.F is not attached, you will get compatibility mode aggregate A.B.C mounted.

The maximum size of the file system you just created is known as its **quota**. This is a logical number that is compared against each time additional blocks are allocated to the file system. When the quota is reached, the file system indicates that it is full (even if there are more physical blocks available in the aggregate). A quota can be smaller than the space available in the aggregate (this is typical), it can be equal or it can be larger. If the quota is larger than the space available in the aggregate, or more typically, if the sum of the quotas for all file systems in an aggregate is larger than the space available in the aggregate, the file system can run out of physical space before it reaches its quota.

The quota of a file system can be displayed by using the `zfsadm lsquota` command and it can be increased by using the `zfsadm setquota` command. The quota of a file system can also be decreased (if the usage has not exceeded the new quota) by using the `zfsadm setquota` command. The quota is the number used when determining if a message to the operator is required because of the `FSFULL` parameter of the `MOUNT` command. See **“MOUNT” on page 109** for more information.

A file system’s quota can be dynamically increased if the `FSGROW PARM` was specified on the `MOUNT` or if the `fsgrow` option is specified in the IOEFSPRM file. You must specify the amount that the quota should grow (in k-bytes) and the number of times that the quota should be increased. (A history of the number of times the quota has been increased is not kept across instances of the zFS PFS. That is, the number of times the quota has been increased is lost when zFS is stopped and restarted.)
If you attempt to attach an aggregate that contains a duplicate file system name, the attach is successful and a message is issued that states there are two file systems with the same filesystemname in use. On the zfsadm commands that refer to file systems, the -aggregate option can qualify the file system name. MOUNT allows an AGGREGATE PARM.

After creating a zFS file system in a multi-file system aggregate, the file system can be mounted. (The aggregate must be attached before any file systems in a multi-file system aggregate can be MOUNTed.) The following example is for a TSO/E MOUNT command for the zFS file system just created:

MOUNT FILESYSTEM('OMVS.PRV.FS1') MOUNTPOINT('/etc/mountpt2') TYPE(ZFS) MODE(RDWR) NOAUTOMOVE

The previous example assumes that the directory /etc/mountpt2 exists and is available to become a mount point. Note that the TYPE parameter of the MOUNT command specifies ZFS. This is required for any zFS file systems in multi-file system aggregates. After the zFS file system is mounted, applications and commands can be executed and files and directories can be accessed in zFS just as in HFS. Chapter 5, “Using zFS in a shared file system environment,” on page 31 explains why the NOAUTOMOVE is specified for file systems in multi-file system aggregates.

When multiple file systems are created in an aggregate, this allows the possibility of space sharing between those file systems. That is, physical DASD space that is made available by erasing files in one file system (A), is potentially available to another file system (B) in the same aggregate (assuming that the other file system (B) is not at its quota limit).

Starting with z/OS V1R7, when a zFS multi-file system aggregate is attached R/W, the on-disk format of the aggregate is modified. It is changed from a version 1.3 aggregate to a version 1.4 aggregate. This allows the performance of mount to be improved (especially for zFS file systems with many files and directories). During the automatic conversion, you will see messages such as:

IOEZ00500I Converting PLEX.JMS.AGGR007.LDS0007 for fast mount processing
IOEZ00518I Converting filesystem PLEX.JMS.AGGR007.LDS0007 to allow for fast mount

You must install toleration APAR OA11573 on prior releases so that prior releases can correctly access the new structure (version 1.4) for ZFS aggregates. If you do not install toleration APAR OA11573 on prior releases, prior releases will not be able to correctly access the new structure.

In this case, you can convert a ZFS aggregate back to a version 1.3 structure so that it can be accessed. (You should, of course, apply toleration APAR OA11573 as soon as possible.) To convert a zFS aggregate back to a version 1.3 structure, use the ZFS IOEAGSLV (salvager) utility. A new option (-converttov3) is provided to convert a version 1.4 ZFS aggregate back to a version 1.3 ZFS aggregate. The IOEAGSLV utility (with the new -converttov3 option) is provided with z/OS Version 1 Release 8. IOEAGSLV is also installed in the MIGLIB PDS. IOEAGSLV can be executed from any supported release by STEPLIBing to MIGLIB. Here is a sample job:

```
//USERIDA JOB ,'Salvage',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//STELIB DD DSN=hlq.MIGLIB,DISP=OLD
//SALVAGE EXEC PGM=IOEAGSLV,REGION=0M,
// PARM=('aggregate PLEX.JMS.AGGR007.LDS0007 -converttov3')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*
```
Growing a multi-file system aggregate

If the sum of the quotas of all the file systems in an aggregate is greater than the physical space available in the aggregate, it is possible for a file system to run out of physical space before exceeding its quota. If this occurs, the application gets ENOSPC as a return code (the same return code it gets for exceeding its quota). 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 on the volume to extend the aggregate’s VSAM LDS. The size specified on the `zfsadm grow` command must be larger than the current size of the aggregate.

For example, suppose a 2 cylinder (primary allocation, 3390) aggregate has a total of 180 8K blocks and a (potential) secondary allocation of 1 cylinder. 180 8K blocks is 1440K bytes. A `zfsadm aggrinfo` command for this aggregate might show 1440K. `zfsadm grow` does this by calling DFSMS to allocate the additional DASD space. You might need to specify a few blocks larger than the current size before an allocation occurs because DFSMS can require some number of reserved blocks. Here is an example:

```
zfsadm aggrinfo omvs.prv.aggr004.lds0004

OMVS.PRIV.AGGR004.LDS0004 (R/W MULT): 1295 K free out of total 1440
zfsadm grow omvs.prv.aggr004.lds0004 1440

IOEZ000173I Aggregate OMVS.PRIV.AGGR004.LDS0004 successfully grown
OMVS.PRIV.AGGR004.LDS0004 (R/W MULT): 1295 K free out of total 1440
zfsadm grow omvs.prv.aggr004.lds0004 1441

OMVS.PRIV.AGGR004.LDS0004 (R/W MULT): 2015 K free out of total 2160
```

The aggregate now has a total size of 2160K bytes. The size of the aggregate is rounded up to the control area (CA) size. You can specify 0 for the size to get a secondary allocation size extension. In this case, a secondary allocation must have been specified on the VSAM LDS. File systems that have not exceeded their quota can now use the additional physical space that is available. (If necessary, a file system quota can be increased with the `zfsadm setquota` command.) Aggregates cannot be made smaller.

Dynamically growing a multi-file system aggregate

An administrator can specify that an aggregate should be dynamically grown if it becomes full. This is specified by the `-aggrgrow` option on the `zfsadm attach` command or the `aggrgrow` suboption of the `define_aggr` option of the IOEFSPRM file or globally by the `aggrgrow` option of the IOEFSPRM file. 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 aggregate will be extended when an operation cannot complete because the aggregate is full. If the extension is successful, the operation will be redriven transparently to the application.

When an aggregate or file system becomes full

When a zFS file system in a multi-file system aggregate becomes full, you can add more space. But first, you must first determine if you have run out of physical space in the aggregate or if you have reached the file system quota limit. You can determine how much free space there is in the aggregate by using the `zfsadm aggrinfo` command. For example:

```
zfsadm aggrinfo PLEX.JMS.AGGR001.LDS0001

PLEX.JMS.AGGR001.LDS0001 (R/W MULT): 2964 K free out of total 3600
```

You can determine how full the file system is by using the `zfsadm lsquota` command. For example:

```
zfsadm lsquota -filesystem OMVS.PRIV.FS3 -aggregate PLEX.JMS.AGGR001.LDS0001

Filesys Name    Quota     Used  Percent  Used    Aggregate
OMVS.PRIV.FS3  7000       9     0        17 = 636/3600 (zFS)
```
An aggregate can grow automatically or you can explicitly grow it. Also, a file system quota can grow automatically or you can explicitly set it larger. See the following table:

<table>
<thead>
<tr>
<th>Automatic</th>
<th>Explicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify secondary allocation in DEFINE of VSAM LDS; and</td>
<td>zfsadm grow aggname newsizeinK</td>
</tr>
<tr>
<td>Specify -aggrgrow in IOEFSRM or on zfsadm attach, and</td>
<td></td>
</tr>
<tr>
<td>have space available on volume</td>
<td></td>
</tr>
<tr>
<td>Specify fsgrow(x,y) in IOEFSRM or on MOUNT PARM</td>
<td>zfsadm setquota filesystemname newsizeinK</td>
</tr>
</tbody>
</table>

Another way to relieve a file system full condition is to erase some files. If your aggregate is full and you have cloned file systems in the aggregate, you might be able to free some physical space by recloning one or more of the file systems.

### Comparing compatibility mode aggregates and multi-file system aggregates

The difference between a compatibility mode aggregate and a multi-file system aggregate is the number of read-write file systems in the aggregate and whether the aggregate has been explicitly attached or not. There is no special bit stored on the disk that indicates whether an aggregate is compatibility mode or multi-file system.

A compatibility mode aggregate has exactly one read-write file system in the aggregate and it is not attached before being mounted. It is only mounted and unmounted. (An implicit attach occurs during the mount; an implicit detach occurs during the unmount. If the MOUNT is of type RDWR, the aggregate is attached R/W. If the MOUNT is of type READ, the aggregate is attached R/O unless the RW PARM is specified on the MOUNT.) To mount a compatibility mode aggregate, the aggregate name is specified as the file system name. The decision as to whether to treat an aggregate as a compatibility mode aggregate is made at mount time or explicit attach time. If no attach has been done and the mount is successful, the aggregate is treated as a compatibility mode aggregate. The mount will only be successful if there is exactly one read-write file system in the aggregate. If the name of the (only) file system in the aggregate does not match the aggregate name, the file system name will be renamed to be the same as the aggregate name. For a mount of type RDWR, the file system name will be changed on disk. For a mount of type READ, the changed name will be kept in memory.

If an explicit attach is done before any mounting, the aggregate is treated as a multi-file system aggregate. If you want, you can cause zFS to treat an aggregate that has been formatted with the `-compat` option, as a multi-file system aggregate by attaching it before you mount the file system. You can then create another file system in the aggregate. This, however, means that the aggregate cannot be treated as a compatibility mode aggregate anymore.

You can always query an (attached) aggregate to determine if it is compatibility mode or multi-file system using the `zfsadm aggrinfo` command. COMP indicates compatibility mode; MULT indicates multi-file system.

### Sharing zFS data between systems

The only fully supported mechanisms for sharing zFS data between systems are:

- using shared file systems in a sysplex (see Chapter 5, “Using zFS in a shared file system environment,” on page 31)
- read-only sharing (a zFS aggregate is attached read-only to multiple systems and no system has the aggregate attached read-write)
There is limited support for a zFS aggregate to be attached read-write to one system and attached read-only on another system.

If you attach a zFS aggregate read-write to a system (system A), you cannot attach that aggregate read-write to any other system (system B). If you attempt to attach the aggregate read-write to another system (system B), the attach will fail.

If you subsequently attach the aggregate read-only on another system (system B), the attach will succeed if no data has been written to the aggregate since it was attached read-write. If data has been written, you will receive reason code EFxx6271 indicating that the log must be replayed and the read-only attach will fail.

Notes:
1. If you are running z/OS V1R6 or later on the system where the file system is mounted read-write, you can quiesce and unquiesce the aggregate and clear the log. For example:
   a. First enter, `zfsadm quiesce -aggrname name`
   b. Next, `zfsadm unquiesce -aggrname name`
2. If you do share a read-write and a read-only and have `romount_recovery=on`, a read-only mount that needs to run the log will still fail because when the aggregate is temporarily attached read-write, the block 0 daemon will see that another system has it mounted read-write and fail the mount.

Otherwise, you need to detach the read-write aggregate from the system (system A) and then attach the aggregate read-write on that same system (system A). This will clear the log so that it can be attached read-only (on system B). After the aggregate is successfully attached read-only, you will receive a message (informational message “IOEZ00439I Read-only aggregate `aggrname` is attached read-write on another system.”) and errors might occur on the read-only aggregate (system B) if writes have occurred on the aggregate from the system where it is read-write attached. To recover from the errors on the read-only aggregate, you need to detach the read-write aggregate (system A) and then attach it read-write (system A). Then you can detach the read-only aggregate and then attach it read-only (system B).
Chapter 9. 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. See the topic on “Sharing file systems in a sysplex” in z/OS UNIX System Services Planning for recommendations for the system file systems.

In addition, if a file system is mounted read-write, but accessed mainly from a single 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 when you run all systems with zFS sysplex-aware. Note that normally, when a zFS file system is owned on a system running zFS sysplex-aware and there are other systems running zFS non-sysplex aware, you cannot explicitly move z/OS UNIX ownership from the zFS sysplex-aware system to the zFS non-sysplex aware system. In that case, you need to unmount it and then mount it on the zFS non-sysplex aware system. See the discussion on “Running mixed zFS sysplex-aware and zFS non-sysplex aware” on page 13.

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 there are no volumes or channels that are pushed beyond their capacity. The following information describes 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.

You might see messages IOEZ00188A, IOEZ00662I, or IOEZ00663I indicating 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” on page 58
- “Transaction cache” on page 58
- “Vnode cache” on page 58
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 which includes all directory contents, file status information (such as atime, mtime, size, permission bits, and so on), file system structures and 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 additional information about how zFS shows free blocks, see [zFS disk space allocation](#) on page 25.

- **inline** If the file is smaller than 52 bytes, its data is stored in the structure that contains the status information for the file.
- **fragmented** If the file is less than 7 K, it is stored in blocks on disk that can be shared with other files, hence multiple files are stored in the same physical disk block. Physical disk blocks are always 8 K in size.
- **blocked** 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 the same 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 very 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 32 M. Because the metadata cache only contains 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.

**Transaction cache**

Every change to zFS file system metadata is bounded by a transaction describing 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, there is no need for the administrator to tailor the transaction cache size. However, the `MODIFY ZFS,QUERY,ALL` output will show 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 and recycles these vnodes 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 file’s status 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 32768. 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 essentially 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 only requires 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 K or larger. For files smaller than 7 K, I/O is typically performed through the metadata cache. Starting in V1R11 when you run sysplex-aware, zFS also has a client cache file used for file caching on client systems.

The user file and client cache are allocated in data spaces. The sizes by default are 256M for the user_cache_size and 128M for the client_cache_size (only meaningful when zFS is running sysplex-aware) and can be tailored to meet your performance needs based on your overall system memory. The maximum size is 65536 M (64 G). The general rule for any cache is to ensure a good hit ratio. Additionally, for a user file cache it is good to have it 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 exported exclusively to SMB clients by using the SMB server will likely have a low hit ratio because the SMB client and the SMB server caches data, making the zFS cache achieve a low hit ratio. This is expected and is not considered a problem.

**NOREADAHEAD option**

For sequential file access, read-ahead provides an overlap of I/O with processing that can result in smaller response time for file read requests. However for random file access, read-ahead can degrade performance. zFS generally attempts to first determine if a file’s access pattern is sequential or random before it decides if read-ahead should be performed for that file. You must not specify this option unless under the direction of IBM service.

**Log files**

Every zFS aggregate contains a log file used to record transactions describing changes to the file system structure. This log file is, by default, 1% of the aggregate size, but you can tailor it on the ioegfmt 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 K buffers used to contain log file updates. You must not modify this file unless under the direction of IBM service. Log file buffers are always written asynchronously to disk and typically only need to be waited upon 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 is 16 M. The log file cache is grown dynamically by adding one 8 K buffer for each attached aggregate. This ensures each aggregate always has one 8 K 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, but the MODIFY ZFS,QUERY,ALL command shows log file I/O waits, therefore make the log file cache large enough to avoid frequent 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, and the log file cache by choosing the fixed option for the cache. This ensures the cache experiences no paging and avoids page fixing for each I/O. This does come at the expense of using real storage for the given 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, you should not set it.

I/O balancing

Any file system’s performance 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™ will show DASD performance.

zFS operator MODIFY ZFS,QUERY,ALL 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 easily balance I/O among your DASD. For example, you can use the QUERY 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 syntax of this command is: MODIFY ZFS,QUERY,<report>,<option>

where <report> is:

- **KN** This report provides counts of calls made to zFS from z/OS UNIX and the average response time of each call. This is the basic measure of zFS performance. There are no <option>s for this report. See "KN" on page 61 for details of the report.

- **VM** This report provides performance information for the user file cache including cache hit ratios, I/O rates and storage usage. There are no <option>s for this report. See "VM" on page 63 for details of the report.

- **LFS** This report provides detailed file system statistics including the performance of the zFS metadata caches, the vnode cache and the aggregate I/O statistics. There are no <option>s for this report. See "LFS" on page 65 for details of the report.

- **LOCK** This report provides a measure of how much lock contention and how often z/OS UNIX threads wait for certain events such as user file cache reclaim. There are no <option>s for this report. See "LOCK" on page 69 for details of the report.

- **STOR** This report provides a detailed breakdown of zFS allocated storage by component. By default this
This report just lists storage usage by zFS component, if you use the details option then you will get more detailed information for each zFS component. See "STOR" on page 70 for details of the report.

**FILE**  This report provides a detailed breakdown of requests per zFS files 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 71 for details of the report.

**STKM**  This report displays the current server token manager (STKM) statistics. See "STKM" on page 72 for details of the report.

**CTKC**  This report displays the client token manager statistics. CTKC is only present when the system is a Sysplex client of another system. See "CTKC" on page 73 for details of the report.

**IOBYDASD**  This report 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 72 for details of the report.

**SVI**  This report displays the server vnode interface (SVI) calls from other systems to this server. See "SVI" on page 74 for details of the report. SVI will be present when the system is a server to another system.

**ALL**  This report shows all the above reports. However, for the STOR report, the details option is off and the FILE report indicates only active file systems.

You can also reset the statistics for any given zFS report or reset all of the internal zFS statistics. The syntax of this command is:

```
MODIFY ZFS,RESET,<report>
```

where <report> is KN, VM, LFS, LOCK, STOR, FILE, STKM, CTKC, SVI, ALL.

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 p.m. and 3 p.m. you enter MODIFY ZFS,RESET,ALL at 1 p.m. and enter MODIFY ZFS,QUERY,ALL at 3 p.m.:

```
MODIFY ZFS,RESET,ALL

MODIFY ZFS,QUERY,ALL
```

Monitor period

```
MODIFY ZFS,QUERY,ALL
```

The zFS query output from the MODIFY ZFS,QUERY, <report> command is written to the system log.

**Sample zFS query reports**

The next sections show sample zFS query outputs and describes the relevant fields of each report. Some fields are used mainly by IBM service but are included here for completeness.

**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 and the average response time in milliseconds for each request. These requests are the official interface between z/OS UNIX and zFS and is the most fundamental measure of zFS performance because it includes any CPU, I/O wait time, or lock wait time.

The times here are 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.
Note: If you see abnormally long times listed for `zfs_lookup`, `zfsCreates`, or `zfsRemoves`, 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 28.

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>123636</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_ioctlS</td>
<td>66</td>
<td>0</td>
<td>0.685</td>
</tr>
<tr>
<td>zfs_getattrS</td>
<td>17561</td>
<td>1</td>
<td>0.033</td>
</tr>
<tr>
<td>zfs_setattrS</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>zfs_creates</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_symlinks</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>
<tr>
<td>zfs_fsyncS</td>
<td>188</td>
<td>0</td>
<td>9.310</td>
</tr>
<tr>
<td>zfs_truncS</td>
<td>553</td>
<td>0</td>
<td>3.278</td>
</tr>
<tr>
<td>zfs_lockctls</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>zfs_audits</td>
<td>357</td>
<td>0</td>
<td>0.051</td>
</tr>
<tr>
<td>zfs_inactives</td>
<td>2347</td>
<td>5</td>
<td>0.013</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>85</td>
<td>13</td>
<td>6.528</td>
</tr>
<tr>
<td>zfs_statfss</td>
<td>245</td>
<td>0</td>
<td>0.278</td>
</tr>
<tr>
<td>zfs_mnts</td>
<td>22</td>
<td>0</td>
<td>435.707</td>
</tr>
<tr>
<td>zfs_umnts</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* 334045 1966 1.125

F ZFS,QUERY,KNPFS
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>2.665</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_ioctlS</td>
<td>15</td>
<td>0</td>
<td>0.096</td>
</tr>
<tr>
<td>zfs_getattrS</td>
<td>14560</td>
<td>3972</td>
<td>1.170</td>
</tr>
<tr>
<td>zfs_setattrS</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_lookups</td>
<td>47122</td>
<td>1606</td>
<td>0.340</td>
</tr>
<tr>
<td>zfs_createS</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_renameS</td>
<td>1151</td>
<td>1151</td>
<td>5.673</td>
</tr>
<tr>
<td>zfs_mkdirS</td>
<td>603</td>
<td>603</td>
<td>6.548</td>
</tr>
<tr>
<td>zfs_rmdirS</td>
<td>582</td>
<td>582</td>
<td>5.834</td>
</tr>
<tr>
<td>zfs_readdirS</td>
<td>317</td>
<td>27</td>
<td>3.625</td>
</tr>
<tr>
<td>zfs_symlinkS</td>
<td>55</td>
<td>55</td>
<td>8.031</td>
</tr>
<tr>
<td>zfs_readlinkS</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>
</tbody>
</table>

62 z/OS V1R11.0 Distributed File Service zFS Administration
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. This size of both of these caches are 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 dataspaces when it can to reduce lock contention (as shown in this example). zFS has a structure for each file currently cached, each cached file is broken into 64K segments and each segment is broken into 4K pages. A segment is assigned to a dataspaces, hence the pages for any given segment belong only to one dataspaces. A file's segments 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 LRU fashion.

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 64K).

```
IOEZ00438I Starting Query Command VM.
User File (VM) Caching System Statistics
---------------------------
External Requests:
Reads 1 Fsyncs 0 Schedules 3
Writes 30000 Setattrs 1 Unmaps 0
Asy Reads 1 Getattrs 9 Flushes 0

File System Reads:
Reads Faulted 1 (Fault Ratio 100.00%)
Writes Faulted 0 (Fault Ratio 0.00%)
Read Waits 0 (Wait Ratio 0.00%)
Total Reads 1

File System Writes:
Scheduled Writes 96 Sync Waits 0
Error Writes 0 Error Waits 0
Scheduled deletes 0
Page Reclaim Writes 0 Reclaim Waits 0
Write Waits 0 (Wait Ratio 0.00%)
```

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

---

Chapter 9. Performance and debugging 63
Sysplex Client Caching System Statistics

---

External Requests:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reads</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Writes</td>
<td>46737</td>
<td></td>
</tr>
<tr>
<td>Asy Reads</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Fsyncs</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Schedules</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Unmaps</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Getattrs</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Flushes</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

File System Reads:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reads Faulted</td>
<td>0</td>
<td>(Fault Ratio 0.00%)</td>
<td></td>
</tr>
<tr>
<td>Writes Faulted</td>
<td>0</td>
<td>(Fault Ratio 0.00%)</td>
<td></td>
</tr>
<tr>
<td>Read Waits</td>
<td>0</td>
<td>(Wait Ratio 0.00%)</td>
<td></td>
</tr>
<tr>
<td>Total Reads</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

File System Writes:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduled Writes</td>
<td>1462</td>
<td>Sync Waits</td>
<td>0</td>
</tr>
<tr>
<td>Error Writes</td>
<td>106</td>
<td>Error Waits</td>
<td>0</td>
</tr>
<tr>
<td>Scheduled deletes</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page Reclaim Writes</td>
<td>0</td>
<td>Reclaim Waits</td>
<td>0</td>
</tr>
<tr>
<td>Write Waits</td>
<td>0</td>
<td>(Wait Ratio 0.00%)</td>
<td></td>
</tr>
</tbody>
</table>

Page Management (Segment Size = 32K) (Page Size = 4K)

---

External Requests: This section of the report describes the requests made to the user file cache to perform operations as requested by applications.

- **Reads, Writes**: how often the cache was called to read or write files.
- **Asy Reads**: how often read-ahead is performed.
- **Fsync**: how often applications requested that zFS sync a file's data to disk.
- **Unmaps**: the count of file deletions.

File System Reads: This section 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.

- **Reads Faulted**: the count of read requests that needed to perform at least one I/O to read the requested portion of the file from disk.
• Writes Faulted - the 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 needs to 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.
• Read Waits - 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).
• Total Reads - the total number of file system reads made for any reason.

File System Writes: This section 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.
• Scheduled Writes - the count of how often the cache wrote out dirty segments for a file. Segments are written as soon as every page becomes dirty. 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).
• Sync Waits - the count of how often an fsync request needed to wait on pending I/O for dirty segments.
• Error Writes and Error Waits - 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. (Note that a file system 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).
• Scheduled Deletes - the count of times a pending I/O was cancelled because of a file being deleted. In this case, the data is not appropriate to be on disk (because the file is 0 link count) and therefore cancelling the I/O is done to avoid an I/O wait. This is a performance optimization for file remove.
• Page Reclaim Writes - the count of times that a segment had to be written to DASD to reclaim space in the cache.
• Page Reclaim Waits - the count of times that the reclaim function waited on pending I/O to reclaim segment pages.
• Write Waits - the count of times a write occurred to a page that was already pending I/O. In this case, the I/O needs to be waited upon before the page is updated with the new data.

Page Management: This section of the report shows the user file cache storage use. It shows total pages, number of free pages, and total number of segments. Each dataspace used to hold cache pages is shown with the total number of pages and number of free pages and allocated segments.

LFS

LFS:

<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>114</td>
</tr>
<tr>
<td>efs_rele</td>
<td>0</td>
<td>efs_create</td>
<td>307</td>
</tr>
<tr>
<td>efs_inactive</td>
<td>0</td>
<td>efs_remove</td>
<td>308</td>
</tr>
<tr>
<td>efs_getattr</td>
<td>2926</td>
<td>efs_rename</td>
<td>13</td>
</tr>
<tr>
<td>efs_setattr</td>
<td>17</td>
<td>efs_mkdir</td>
<td>27</td>
</tr>
<tr>
<td>efs_access</td>
<td>6772</td>
<td>efs_rmdir</td>
<td>27</td>
</tr>
<tr>
<td>efs_lookup</td>
<td>2303</td>
<td>efs_link</td>
<td>1</td>
</tr>
<tr>
<td>efs_getvolume</td>
<td>0</td>
<td>efs_symlink</td>
<td>4</td>
</tr>
<tr>
<td>efs_getlength</td>
<td>0</td>
<td>efs_readlink</td>
<td>5</td>
</tr>
<tr>
<td>efs_afsfid</td>
<td>0</td>
<td>efs_rdwr</td>
<td>0</td>
</tr>
<tr>
<td>efs_fid</td>
<td>0</td>
<td>efs_fsync</td>
<td>0</td>
</tr>
<tr>
<td>efs_vmrread</td>
<td>6</td>
<td>efs_waitIO</td>
<td>4439</td>
</tr>
<tr>
<td>efs_vmrwrite</td>
<td>4595</td>
<td>efs_cancelIO</td>
<td>4268</td>
</tr>
<tr>
<td>efs_clrsetid</td>
<td>0</td>
<td>efs_audit</td>
<td>0</td>
</tr>
<tr>
<td>efs_atime</td>
<td>0</td>
<td>efs_vmblkinfo</td>
<td>131</td>
</tr>
</tbody>
</table>

Total zFS Vnode Ops: 26263
## 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>150000</td>
<td>604623</td>
<td>241</td>
<td>0.39%</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

zFS Vnode structure size: 216 bytes
zFS extended vnodes: 150000, extension size 692 bytes (minimum)
Held zFS vnodes: 4 (high 121)
Open zFS vnodes: 0 (high 117)
Reusable: 149996

## Metadata Caching Statistics

<table>
<thead>
<tr>
<th>Buffers (K bytes)</th>
<th>Requests</th>
<th>Hits</th>
<th>Ratio</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>4224</td>
<td>33792</td>
<td>139453</td>
<td>104957</td>
<td>75.2%</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>4096</td>
<td>32768</td>
<td>340</td>
<td>0.0%</td>
<td>0</td>
</tr>
</tbody>
</table>

## Directory Cache Statistics

<table>
<thead>
<tr>
<th>Dir Blocks (K bytes)</th>
<th>Requests</th>
<th>Hits</th>
<th>Ratio</th>
<th>Deletes</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>2048</td>
<td>3961</td>
<td>99.318%</td>
<td>27</td>
</tr>
</tbody>
</table>

## Transaction Cache Statistics

Transactions started: 11422  Lookups on tran: 238405  EC Merges: 1195
Allocated Transactions: 2000  (Act= 0, Pend= 0, Comp= 1456, Free= 544)

## 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>177</td>
<td>209</td>
<td>0</td>
<td>0</td>
<td>File System Metadata</td>
</tr>
<tr>
<td>1418</td>
<td>197</td>
<td>0</td>
<td>1027</td>
<td>Log File</td>
</tr>
<tr>
<td>4595</td>
<td>150</td>
<td>1727</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>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>Metadata cache read</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>User file cache direct read</td>
</tr>
<tr>
<td>0</td>
<td>0</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>148</td>
<td>68</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>4440</td>
<td>77</td>
<td>1727</td>
<td>0</td>
<td>User File cache direct write</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Metadata cache file sync write</td>
</tr>
<tr>
<td>78</td>
<td>51</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>97</td>
<td>156</td>
<td>0</td>
<td>0</td>
<td>Metadata cache file system quiesce write</td>
</tr>
<tr>
<td>1418</td>
<td>197</td>
<td>0</td>
<td>1027</td>
<td>Log file write</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Metadata cache shutdown write</td>
</tr>
</tbody>
</table>
zFS I/O by Currently Attached Aggregate

<table>
<thead>
<tr>
<th>DASD</th>
<th>VOLSER</th>
<th>PAV</th>
<th>IOs</th>
<th>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>PRV001</td>
<td>1</td>
<td>R/W</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SUDFS5.PRIVATE.LFSNET</td>
<td></td>
</tr>
<tr>
<td>PRV001</td>
<td>1</td>
<td>R/W</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SUDFS5.PRIVATE.LFS106</td>
<td></td>
</tr>
<tr>
<td>PRV002</td>
<td>1</td>
<td>R/W</td>
<td>9</td>
<td>188</td>
<td>3427</td>
<td>205608</td>
<td>0</td>
<td>SUDFS5.PRIVATE.LFSFS</td>
<td></td>
</tr>
<tr>
<td>PRV001</td>
<td>1</td>
<td>R/W</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SUDFS5.PRIVATE.TESTGROW</td>
<td></td>
</tr>
</tbody>
</table>

Summary:

Total number of waits for I/O: 556
Average I/O wait time: 62.215 (msecs)

**zFS Vnode Op Counts:** This section 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.

**zFS Vnode Cache Statistics:** This section shows the zFS vnode cache statistics. It shows the number of currently allocated vnodes and the vnode hit ratio. "Allocates" and "Deletes" 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. 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 there are additional data structures anchored from the vnode that take space. Everything added together yields over 1K 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.

**Metadata Caching Statistics:** This section 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 7K in size. For files smaller than 7K, zFS will place multiple files in one disk block (for zFS a disk block is 8K 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 8K in size), the total bytes, the request rates and hit ratio of the cache. 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.

**Metadata Backing Cache Statistics:** This section 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 example above the metadata backing cache is providing no
performance benefit (as shown by its 0 hit ratio). The metadata backing cache is not created by default. It can only be created by specifying the metaback_cache_size configuration option of the IOEFSprm file or the \texttt{zfsadm config} command.

\textbf{Directory Cache Statistics:} zFS maintains a cache of directory buffers. This directory cache is also backed by the metadata cache (that is, a directory page is always read to/from the metadata cache into/out of the directory cache). The size of this cache cannot be controlled by the administrator and typically does not have any performance ramifications. For details, see the \texttt{dir_cache_size} in \texttt{IOEFSprm} on page 166.

\textbf{Transaction Cache Statistics:} 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 (Active, Pending, Complete or Committed):

\begin{itemize}
  \item \textbf{Active} There are still records being written to the log file describing updates being made by this transaction, hence the transaction was started but has not yet ended. (This is shown as “Act” in the report.)
  \item \textbf{Complete} 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. (This is shown as “Comp” in the report.)
  \item \textbf{Committed} 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 report statistics for this count is not shown. As soon as a transaction is committed the structure representing the transaction is “free” for reuse for another transaction.)
  \item \textbf{Equivalence Classes} zFS does not use a common technique called 2 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 thus 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 “EC Merges”.
  \item \textbf{Pending} 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. (This is “Pend”.)
\end{itemize}

The transaction cache size is by default 2000 transactions. It can be changed by the \texttt{tran_cache_size} configuration option. In general, zFS will increase 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 \texttt{zfsadm config} command to set the \texttt{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.
I/O Summary By Type & Circumstance: This section is mainly for IBM internal use in diagnosing performance related problems. zFS keeps detailed statistics on how often it performs I/O for various circumstances and how often it waits on that I/O for determination of performance problems.

zFS I/O by Currently Attached Aggregate: 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. (Example, for a PAV DASD with five paths will issue, at the most, three I/Os at one time to Media Manager).

The reason zFS limits the I/O is that 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, and 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 cancelled. For example, this is done in cases where a file was written, and then immediately deleted. Finally, the zFS I/O driver will merge 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.

The "PAV I/Os" column shows how many I/Os are sent in parallel to Media Manager by zFS, non PAV DASD always shows the value 1.

The DASD VOLSER for the primary extent of each aggregate is shown along with the total number of I/Os and bytes read/written.

Finally, the number of times a thread processing a request must wait on I/O and the average wait time in milliseconds is shown. By using this information in conjunction 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’s affected by it but it’s 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).

This report along with RMF DASD reports and the zFS FILE report can be used to balance zFS aggregates among DASD volumes to ensure an even I/O spread.

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 and lock wait time and sleep waits and sleep wait time can be used in conjunction 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. See the following example:

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)

Top 15 Most Highly Contended Locks

<table>
<thead>
<tr>
<th>Thread</th>
<th>Async Wait</th>
<th>Spin Disp.</th>
<th>Spin 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>30.285%</td>
<td></td>
<td>Anode bitmap allocation handle</td>
</tr>
<tr>
<td>481</td>
<td>0</td>
<td>28</td>
<td>10.249%</td>
<td></td>
<td>Anode fileset quota lock</td>
</tr>
<tr>
<td>291</td>
<td>0</td>
<td>42</td>
<td>6.705%</td>
<td></td>
<td>Transaction lock</td>
</tr>
<tr>
<td>205</td>
<td>0</td>
<td>62</td>
<td>5.376%</td>
<td></td>
<td>Metadata-cache buffer lock</td>
</tr>
<tr>
<td>210</td>
<td>0</td>
<td>4</td>
<td>4.309%</td>
<td></td>
<td>Anode fileset handle lock</td>
</tr>
<tr>
<td>84</td>
<td>68</td>
<td>7</td>
<td>3.201%</td>
<td></td>
<td>User file cache main segment lo</td>
</tr>
<tr>
<td>0</td>
<td>55</td>
<td>0</td>
<td>1.107%</td>
<td></td>
<td>Volser I/O queue lock</td>
</tr>
<tr>
<td>38</td>
<td>0</td>
<td>0</td>
<td>0.765%</td>
<td></td>
<td>Vnode-cache access lock</td>
</tr>
<tr>
<td>12</td>
<td>23</td>
<td>11</td>
<td>0.724%</td>
<td></td>
<td>Transaction-cache main lock</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>3</td>
<td>0.443%</td>
<td></td>
<td>Transaction-cache equivalence c</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0.422%</td>
<td></td>
<td>Async IO event lock</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0.281%</td>
<td></td>
<td>Cache Services association main</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0.100%</td>
<td></td>
<td>Cache Services hashtable resize</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>100.0%</td>
<td>Transaction allocation wait</td>
</tr>
<tr>
<td>0</td>
<td>0.0%</td>
<td>OSI cache item cleanup wait</td>
</tr>
<tr>
<td>0</td>
<td>0.0%</td>
<td>Directory Cache Buffer Wait</td>
</tr>
<tr>
<td>0</td>
<td>0.0%</td>
<td>User file cache Page Wait</td>
</tr>
<tr>
<td>0</td>
<td>0.0%</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)).

By extrapolation, you can guess that the remaining time is CPU time and processor wait time.

STOR
The STOR report provides a breakdown of zFS storage usage. It 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).
Not shown here is the output of QUERY,STOR,DETAILS. That report breaks down each component and shows how much storage is used for each data structure class and is intended primarily for IBM service.

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
--------------------------
Bytes Allocated No. of No. of
Pieces Allocs Frees Component
---------- ------ ------ ------ ---------
67348 18 18 0 Interface
46824 29 118 89 Media Manager I/O driver
71304964 5 5 0 Trace Facility
401876 3 2428 2425 Message Service
318621 1212 1218 6 Miscellaneous
29440 88 118 30 Aggregate Management
113512 112 150 38 Filesystem Management
33096 23 74 51 Administration Command Handling
15472600 98355 98363 8 Vnode Management
15984596 33498 34162 664 Anode Management
34366108 96 96 0 Directory Management
478752 6202 6227 25 Log File Management
34482608 12580 12605 25 Metadata Cache
420312 4014 4014 0 Transaction Management
176082 467 467 0 Asynchronous I/O Component
65724 1187 1199 12 Cache Services
194304 468 468 0 Threading Services
62152 1187 1199 12 Cache Services
11092824 35506 35506 0 User File Cache
53432 90 2213 2123 Storage Management
42344784 3634 5793 2159 XCF Services
28584 16 26 10 Cross system attach validation
2462080 20529 20552 23 Server Token Manager (STKM)
14032 46 46 0 Server Token Cache (STKC)
4697362 115 116 1 Client Token Cache (CTKC)
0 0 0 0 Server Vnode Interface (SVI)
408 2 12 10 Name Space (NS)

FILE
The FILE report lists every file system that was active since the last reset by default (it will list all file systems if you use the ALL option). To conserve space the internal aggregate number is shown rather than the name of the aggregate that contains the file system. Use the zfsadm lsfs command to determine the aggregate name for each file system. The file systems are grouped in the report by aggregate with the most active file systems listed first.

The Flg column indicates the aggregate status as attached (A) and either mounted (M), quiesced (Q), cloning (C), deleting a clone (D), or growing (G). 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 cloning (C) while the other (zFS client) systems can flag it as quiesced (Q). This is because an aggregate that is cloning is quiesced on all other systems and similarly for deleting and growing.
The **Operations** column 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 and aggregates among disks to provide a more even I/O spread.

**FILE:**

<table>
<thead>
<tr>
<th>File System Name</th>
<th>Aggr #</th>
<th>Flg</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMVS.ZFS.DFBLD.DFSSRC</td>
<td>100008</td>
<td>AM</td>
<td>274472</td>
</tr>
<tr>
<td>OMVS.ZFS.LOCAL</td>
<td>100009</td>
<td>AM</td>
<td>111722</td>
</tr>
<tr>
<td>OMVS.ZFS.DCEDFBLD.DCES390.ETC.DCE</td>
<td>100010</td>
<td>AMQ</td>
<td>81632</td>
</tr>
<tr>
<td>OMVS.ZFS.DCEDFBLD.DFSLOCAL</td>
<td>100012</td>
<td>AM</td>
<td>52154</td>
</tr>
<tr>
<td>OMVS.ZFS.DCEDFBLD.OS390R10.ETC</td>
<td>100004</td>
<td>AM</td>
<td>44108</td>
</tr>
<tr>
<td>OMVS.ZFS.GPLTOOLS</td>
<td>100006</td>
<td>AM</td>
<td>8458</td>
</tr>
<tr>
<td>OMVS.ZFS.BLLTOOLS</td>
<td>100007</td>
<td>AM</td>
<td>8120</td>
</tr>
<tr>
<td>OMVS.ZFS.DCEDFBLD.VAR</td>
<td>100005</td>
<td>AM</td>
<td>314</td>
</tr>
<tr>
<td>OMVS.ZFS.USR.LOCAL</td>
<td>100011</td>
<td>AM</td>
<td>54</td>
</tr>
</tbody>
</table>

**IOBYDASD**

The **IOBYDASD** report lists the currently attached DASD by volume. This report is important for viewing the average wait time per I/O.

<table>
<thead>
<tr>
<th>DASD PAV</th>
<th>VOLSER</th>
<th>IOs</th>
<th>Reads</th>
<th>K bytes</th>
<th>Writes</th>
<th>K bytes</th>
<th>Waits</th>
<th>Average Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZY1111</td>
<td>1</td>
<td>67</td>
<td>1252</td>
<td>0</td>
<td>0</td>
<td>64</td>
<td>2.887</td>
<td></td>
</tr>
<tr>
<td>CFC000</td>
<td>1</td>
<td>4</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1.020</td>
<td></td>
</tr>
<tr>
<td>CFC002</td>
<td>1</td>
<td>4</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1.182</td>
<td></td>
</tr>
<tr>
<td>CFC001</td>
<td>1</td>
<td>11</td>
<td>68</td>
<td>2873</td>
<td>11492</td>
<td>12</td>
<td>0.873</td>
<td></td>
</tr>
</tbody>
</table>

Total number of waits for I/O: 84
Average wait time per I/O: 2.429

**STKM**

The **STKM** report lists the server token manager statistics.

<table>
<thead>
<tr>
<th>Server Token Manager (STKM) Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum tokens: 30724</td>
</tr>
<tr>
<td>Tokens In Use: 27687</td>
</tr>
<tr>
<td>Token obtains: 3542592</td>
</tr>
<tr>
<td>Token revokes: 1309562</td>
</tr>
<tr>
<td>Garbage Collects: 666</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>Tokens</th>
<th>Usage Per System:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Obtains</td>
</tr>
<tr>
<td>NP1</td>
<td>3781</td>
<td>897812</td>
</tr>
<tr>
<td>NP2</td>
<td>15147</td>
<td>1233561</td>
</tr>
<tr>
<td>NP3</td>
<td>3</td>
<td>912</td>
</tr>
<tr>
<td>NP4</td>
<td>8756</td>
<td>1410737</td>
</tr>
</tbody>
</table>

**Maximum tokens:** This section lists the token limit at the server. The server runs garbage collection to ensure token maximum not exceeded. In some cases, the system workload might cause the token maximum to be exceeded, such as when there are many open files.
Allocated tokens: The number of tokens allocated in server memory.

File structures: The number of file structures.

Tokens In Use The number of tokens currently held by all clients and the local system.

Token obtains The total number of token obtains by all clients and local system.

Token revokes The total number of token revokes by all clients and local system.

Token returns The total number of token returns by all clients and local system.

Async Grants The 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.

Garbage collects The number of garbage collections of tokens. Garbage collection used to keep the total number of client/local system tokens below the maximum whenever possible.

The following usage example shows the individual counts per system. LOCALUSR is the local system (the server). ZEROLINK is a “special client” used to handle zero link count files and vnode inactivations.

<table>
<thead>
<tr>
<th>Usage Per System:</th>
<th>Tokens</th>
<th>Obtains</th>
<th>Returns</th>
<th>Revokes</th>
<th>Async Grnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP1</td>
<td>3781</td>
<td>897812</td>
<td>894887</td>
<td>502842</td>
<td>0</td>
</tr>
<tr>
<td>NP2</td>
<td>15147</td>
<td>1233561</td>
<td>1188354</td>
<td>415917</td>
<td>0</td>
</tr>
<tr>
<td>ZEROLINK</td>
<td>3</td>
<td>912</td>
<td>909</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NP4</td>
<td>8756</td>
<td>1410737</td>
<td>1402062</td>
<td>504757</td>
<td>0</td>
</tr>
</tbody>
</table>

CTKC

The following report example shows the total number of call counts and the average response time in milliseconds of the call to the system indicated (in this case NP1):

<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>6.621</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><em>TOTALS</em></td>
<td>423222</td>
<td>10.162</td>
</tr>
</tbody>
</table>
SVI
The server vnode interface component handles this call. The report 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.

<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>1401717</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>0.527</td>
</tr>
<tr>
<td>FetchDir</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.188</td>
</tr>
<tr>
<td>Lookup</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>Statfs</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><em>TOTALS</em></td>
<td>2520851</td>
<td>12</td>
<td>195751</td>
<td>1.455</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 the following topics:

- "Overview of trace options for zFS"
- "Understanding the salvager utility" on page 75
- "Overview of dumping for zFS" on page 76
- "Understanding zFS messages" on page 77
- "Determining service levels" on page 78
- "Understanding namespace validation and correction" on page 78
- "Understanding delays and hangs in zFS using the zFS hang detector" on page 79

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 recreating 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.

Requirement: If you use the IOEFSPRM option, zFS must be 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 **F ZFS,TRACE,RESET** command.

4. Re-create the problem.

5. Enter the **F 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 PARMLIB member. For more information, see [Chapter 5, “Using zFS in a shared file system environment,” on page 31](#).

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. 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 used by a particular trace depends on how large the **trace_table_size** is and how recently the trace was reset.

   For example, a 32M **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 2048M, but to print the trace to a PDSE you must limit its size to 750M.

IBM service might require you to trace more events. Additional tracing can be specified in two ways:

- 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.

- Add the events to trace dynamically by entering the **MODIFY ZFS,IOEDEBUG** command. IBM specifies the exact statements needed.

As of z/OS V1R11, the zFS trace table is above the 2 GB bar to avoid consuming space in the zFS address space, which is below the bar.

You can also enter the operator **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 stop and attempt to restart.

If you were not able capture the trace, but you have a zFS dump, IBM service can obtain the trace from the dump.

**Understanding the salvager utility**

The salvager (ioeagslv) 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 cancelled, 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, this 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 when it is appropriate to run the salvager utility against a zFS aggregate. Generally, the following situations indicate running the salvager to ensure there is no corruption or to attempt to correct a corruption:

- A zFS aggregate becomes disabled.
  - An internal error has occurred during zFS processing for the aggregate. You can recognize this situation by the presence of message IOEZ00422E. zFS has detected a problem and has disabled the aggregate so that no writes can occur for this aggregate until it is remounted. This action attempts to avoid writing incorrect data that might corrupt the aggregate. Unmount the aggregate and run the salvage utility.
- An I/O error has occurred while accessing the aggregate. zFS has detected a physical I/O error on the device.
  - This is most likely a hardware problem. You can recognize this situation by the presence of message IOEZ00001E or IOEZ00550E. Unmount the aggregate and run the salvage utility.
- A zFS problem occurs during a mount of a zFS aggregate. zFS has 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.

If a quiesce is not done before backup, corruption of the file system can result. See Chapter 6, “Performing a back up of zFS,” on page 43 for recommendations for backing up zFS aggregates.

Run the salvager utility with the -verifyonly option to examine the aggregate structures. If there are no error messages, then the aggregate is not corrupted. If you run the salvager utility with no options, then it will attempt to fix any corruptions that it finds. It might be necessary to run the salvager several times until you receive no error messages.

The salvager utility is not always able to fix a corrupted aggregate.

- If a fundamental aggregate structure is corrupted, salvager will not be able to recover the aggregate.
- If the aggregate is large or has many objects, salvager might not be able to complete successfully. Even when 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.

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.

**Overview of dumping for zFS**

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:

```
IOEZ00337E 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 84). If zFS does stop, zFS attempts to restart after the terminating exception occurs. If the restart is successful, you might need to remount any zFS file systems.

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 74.

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.

**Understanding zFS messages**

Beginning with z/OS V1R11, zFS uses a simpler XCF communications protocol that also includes validation and correction of any inconsistencies in zFS state information between sysplex members. The messages described below (IOEZ00525I - IOEZ00534I) are now unnecessary and consequently are not issued. This causes zFS V1R11 to require toleration support on V1R9 and V1R10. For details, see z/OS Migration, GA22-7499.

Message IOEZ00617I is issued during zFS initialization to indicates whether zFS is running sysplex-aware for read-write file systems (referred to as sysplex file-support), or not (referred to as sysplex admin-only). It also indicates the zFS XCF protocol level being used:

- 3 - the new z/OS V1R11 level
- 2 - the V1R11 compatible level used by zFS V1R9 and zFS V1R10
- 1 - the old XCF protocol level with some additional support that is compatible with level 2 and level 0
- 0 - the unmodified old XCF protocol.

Beginning with z/OS V1R7, zFS administration commands use XCF communications to exchange zFS aggregate and file system information between members of the sysplex. During zFS initialization, zFS must contact each other zFS system that is active in the sysplex group to announce itself to the other members of the group and to receive information about attached aggregates from the other members of the group. You might see messages (if there are other z/OS V1R7 members with zFS active) on the operator console and in the system log such as:

```
10.27.58 DCEIMGVQ  *IOEZ00525I Starting initialization with DCEIMGVN
10.28.03 DCEIMGVQ  *IOEZ00526I Requesting aggregate information from DCEIMGVN
10.28.07 DCEIMGVQ  IOEZ00528I Initialization with DCEIMGVN complete.
```

These messages are written to the operator console and then are deleted (DOMed) when the target system (DCEIMGVN in this case) responds. Sometimes the messages will be deleted from the operator console before they are displayed. In that case, they will not display on the operator console (they are always in the system log). On the target system, you will find messages (in the system log) such as:

```
IOEZ00529I Preparing for initialization with DCEIMGVQ.
IOEZ00530I Ready to initialize with DCEIMGVQ.
IOEZ00532I Sending aggregate information to DCEIMGVQ.
IOEZ00533I Done initializing with DCEIMGVQ.
```

These messages do not display on the operator console (except for IOEZ00530I which can appear on the operator console). If a failure occurs, you might see a failure message on the initializing or the target system. There can be an abend and a dump associated with this failure. The dump should be sent to IBM service.

If the messages on the initializing system (IOEZ00525I or IOEZ00526I) or the message on the target system (IOEZ00530I) do not get deleted from the operator console in a reasonable period of time, there might be a problem with the initializing system or the target system. You should check to see if there is an outstanding WTOR on either system or if there is a hang on either system.
The IOEFSprm msg_output_dsn option can be specified. It specifies the name of a data set that contains any output messages that come from the zFS PFS. This message output data set is only used for zFS initialization messages. This might be helpful for debugging because this data set can be sent to IBM service if needed. The msg_output_dsn 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 pre-allocated 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 will be written to the data set. (They will still go to the system log.) After zFS restart, the msg_output_dsn data set specified is overwritten.

**Determining service levels**

The service level of the zFS physical file system can be determined by examining the messages that occur on the operator's console when zFS initializes as shown in the following example:

```
IOEZ00559I zFS kernel: Initializing z/OS zSeries File System
Version 01.11.00 Service Level 0000000 - HZFS3B0.
Created on Tue Mar 24 19:41:40 EDT 2009.
Address space asid x3F
```

Or use the `modify zfs,query,level` operator command and look for the following message:

```
IOEZ00639I zFS kernel: z/OS zSeries File System
Version 01.11.00 Service Level 0000000 - HZFS3B0.
Created on Tue Mar 24 19:41:40 EDT 2009.
sysplex(file) interface(3)
```

In z/OS V1R11 and above, when zFS is running with sysplex=on in the IOEFSprm configuration file, the sysplex level is (file), and the interface level is (3). Otherwise, the zFS sysplex level is (admin-only).

In addition, the service level of the `zfsadm` command can be determined by using the `-level` option of the `zfsadm` command. For example:

```
zfsadm -level
IOEZ00020I zfsadm: z/OS zSeries File System
Version 01.11.00 Service Level 0000000 - HZFS3B0.
Created on Tue Mar 24 19:43:46 EDT 2009.
```

**Understanding namespace validation and correction**

Beginning in z/OS V1R11, zFS introduces the concept of namespace validation and correction. First, it is important to understand what the namespace is.

Starting with z/OS V1R7, zFS has supported administration commands and APIs that are sysplex wide in a shared file system environment. 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 stored in each zFS member. If zFS validation detects an inconsistency, one or more messages can occur (for example, IOEZ00612I) and the correction phase begins to correct the inconsistency using one of the following actions:
- Updating the inconsistent information
- Automatically remounting a file system
- Restarting the zFS address space on one or more members.

The corrective action is disruptive and might cause one or more applications to receive I/O errors and display message IOEZ00618E through IOEZ00637E. One or more file system mounts might be lost and require explicit mounts. In addition, zFS can take SVC dumps when it detects a name inconsistency, correspondingly do not issue `f zfs,dump,all`.

Beginning in z/OS V1R11, the zFS XCF protocol is simplified. In prior releases, zFS exchanged information about zFS aggregates and file systems among other zFS members of the sysplex even when no file systems were locally mounted on the other systems. In z/OS V1R11, 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 on other sysplex-aware systems, z/OS UNIX keeps mount information hardened in the couple data set. In addition, zFS keeps track of zFS ownership through the use of 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.

**Requirement:** Because of the simplification of the zFS XCF protocol between zFS members, two addition steps must be performed on prior releases (z/OS V1R9 and z/OS V1R10) to work with z/OS V1R11 zFS:
- You must apply toleration APAR OA25026
- zFS on the prior releases must run at `sysplex_admin_level=2` to allow zFS on z/OS V1R11 to enter the shared file system environment.

For more information and migration instructions, see "Two-step APAR procedure for z/OS V1R11" on page 11 and z/OS Migration, GA22-7499.

### 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.

There are several zFS messages that warn of potential problems in the zFS address space that have to do with delays. This section explains the meaning and what, if anything, you should do. If zFS determines there is a true deadlock, zFS initiates dumps of all systems. The system that detected the deadlock will stop and restart zFS in order 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.

This section refers to IOEZ00xxE zFS messages that 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, GRS). 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 (see the next paragraph). Of course, if a request times out, the request fails.

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 resolving a zFS hang

Perform the following steps when a hang condition occurs.

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, enter D OMVS,W to check the state of sysplex messages/waiters. Message IOEZ00547I (hanging XCF request) can indicate an XCF issue. Check any outstanding message that might need a response to determine if a system is leaving the sysplex or not (for example, IXC402D). This might appear to be a zFS hang, until that message gets a response.

   Also look for the following messages:

   **IOEZ00604I or IOEZ00660I**
   - The delay is outside of zFS. Investigate the identified system service.

   **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 significant amount of auxiliary storage, resolve the auxiliary storage shortage problem. If the message persists, continue to determine if there is a zFS problem.

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

   **Note:** 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.

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

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

5. Interrogate 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 `F ZFS,QUERY,THREADS`. If there is a hang, this user task will persist unchanged over the course of this time span. If the information is different each time, there is no hang.

6. Verify that no zFS aggregates are in the `QUIESCED` state by checking their status using the `zfsadm lsaggr` or `zfsadm aggrinfo` command. For example, quiesced aggregates display 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
   IOEZ00370I A total of 1 aggregates are attached.
   SUSVPI.HIGHRISK.TEST (R/W COMP QUIESCED): 35582 K free out of total 36000
   DCESVPI:/home/susvpi/>
   ```

   Resolve the `QUIESCED` state continuing to determine if there is a real hang condition. The hang condition message can remain on the console for up to a minute after the aggregate is unquiesced.

   **Note:** Message IOEZ00581E appears on the zFS owning systems that contains at least one zFS aggregate that is quiesced. There is a time delay between when the aggregate is quiesced and when the message appears. When there are no quiesced zFS aggregates on the system, this message is DOMed. There is also a delay between when the last aggregate is unquiesced and when the message is DOMed. 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. Verify there are no zFS file systems being cloned by checking their status using the `zfsadm lsfs -long` command. For example, cloning file systems display as follows:

   ```
   # zfsadm lsfs -aggregate PLEX.JMS.AGGR004.LDS0004 -long
   IOEZ001291 Total of 2 file systems found for aggregate PLEX.JMS.AGGR004.LDS0004
   PLEX.JMS.AGGR004.LDS0004 100003,,6 RW (Mounted R/W) states 0x10001 (Clone running)
   4294967232 K alloc limit; 16 K alloc usage
   5040 K quota limit; 26 K quota usage
   16 K Filesystem Inode Table 14 file requests
   version 1.4
   Creation Tue Jun 11 19:18:04 2002
   Last Update Thu Oct 5 14:27:59 2006

   PLEX.JMS.AGGR004.LDS0004.bak 100003,,5 BK (Not Mounted) states 0x30002 On-line
   4294967232 K alloc limit; 26 K alloc usage
   5040 K quota limit; 26 K quota usage
   16 K Filesystem Inode Table 2 file requests
   version 1.4
   Creation Tue Jun 11 19:17:27 2002
   Last Update Thu Oct 5 14:27:59 2006
   ```
**Note:** Message IOEZ00588E appears on the zFS owning system that contains the cloning file system.

8. Check if any user tasks are hung focusing on the tasks issued by IOEZ00524I or IOEZ00660I. User tasks will 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.

```
MODIFY ZFS,QUERY,THREADS
IOEZ00438I Starting Query Command THREADS.
zFS and z/OS UNIX Tasks
-----------------------------------------------
Recov TCB ASID Stack Routine State
-------- -------- ---- -------- -------- --------
7047EA68 007FF290 0050 08C46000 efsb_mkdir WAITLOCK
  since May 26 2:27:34 2006 Current DSA: 08C46CB8
  wait code location offset=039C rtn=elbb_ReadGeneral
  lock=717E9C8 state=F0753A69 owner=(70753A68 0053 7DFE88)
  lock description=Metadata-cache buffer lock
  ReadLock held for 70A17DB8 state=00000002 00000000
  lock description=User-cache resize lock
  ReadLock held for 713C53D0 state=00000002 00000000
  lock description=Anode handle lock

70448988 007DDA48 0053 08C3E70 agown_takeover_work WAITLOCK
  since May 26 2:28:10 2006 Current DSA: 08C44710
  wait code location offset=00E0 rtn=start_aggr_cmd
  lock=71888C70 state=F047F0E9 owner=(7047F0E8 0053 7DCA68)
  lock description=Aggregate syscall lock

7047F0E8 007DDC68 0053 08C3E70 agown_takeover_work WAITLOCK
  since May 26 2:28:10 2006 Current DSA: 08C49980
  wait code location offset=1480 rtn=internal_assoc_iterate
  lock=704478A0 state=F072B308 owner=(7072B308 0053 7DFA68)
  lock description=Aggregate lock

7047E2D8 007DEBBB 0053 08C3E70 agown_takeover_work WAITLOCK
  since May 26 2:28:10 2006 Current DSA: 08C3E710
  wait code location offset=0000 rtn=start_aggr_cmd
  lock=71888C70 state=F047F0E9 owner=(7047F0E8 0053 7DCA68)
  lock description=Aggregate syscall lock

7047E6D8 007D0000 0053 08C3E70 agown_master WAITLOCK
  since May 26 2:28:40 2006 Current DSA: 08C3C2BB
  wait code location offset=030C rtn=agown_master
  lock=71888C70 state=F047F0E9 owner=(7047F0E8 0053 7DCA68)
  lock description=Aggregate syscall lock

7072B308 007DFA60 0053 08C1DE70 block_zero_daemon IOWAIT
  since May 26 2:28:09 2006

70753A68 007D6BBB 0053 08C3E70 local_sync_daemon IOWAIT
  since May 26 2:27:33 2006
  ReadLock held for 716D4178 state=00000002 00000000
  lock description=Log file cache resize lock

70754D68 007C43B0 0053 08C17E70 comm_daemon RUNNING
  since May 26 2:32:59 2006
```

**Note:** This information is for example purposes only.
9. After you ensure there is a valid hang condition and not a slowdown, obtain the appropriate dumps if none are present. IBM Support must have dumps of zFS, OMVS and the OMVS data spaces for problem resolution. Obtain and save SYSLOG and dumps of zFS, OMVS and the OMVS data spaces using JOBNAME=('OMVS',ZFS'),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, SQA, LSQA, PSA, CSA, GRSQ, TRT, SUM, COUPLE'), DSPNAME=('OMVS'.*), END

**Rule:** You must capture dumps for IBM Support before taking any recovery actions (HANGBREAK, CANCEL).

10. If you know which user task is hung (for example, returned in IOEZ00524I), enter the CANCEL or STOP command to clear that task from the system.

11. Finally, if the previous steps do not clear the hang, do one of the following:

   - Enter the MODIFY ZFS,HANGBREAK command to attempt to break the hang condition. The MODIFY ZFS,HANGBREAK command posts any threads that zFS suspects are in a hang condition with an error and can cause abends and dumps to occur, which you can ignore. After entering the MODIFY ZFS,HANGBREAK command, the hang message can remain on the console for up to one minute. When the MODIFY ZFS,HANGBREAK command completes, it issues message IOEZ00025I. However, IOEZ00025I does not mean the system cleared the hang. Enter FDUMP ZFS,QUERY,THREADS to check the output for indication the hang is clear. It is possible that the MODIFY ZFS,HANGBREAK command can clear the current hang condition only to encounter yet another hang. You might have to enter the MODIFY ZFS,HANGBREAK command several times.

   - Or, if you users are hung in the file system, forcefully unmount the file system by entering the MODIFY ZFS,ABORT command.

If you question the hang condition or if the commands mentioned above do not seem to resolve the situation, contact IBM Support and provide all the dumps and SYSLOG information.

### Storage shortage in zFS

When zFS can no longer obtain sufficient storage to complete a request, it issues IOEZ00188I and possibly creates a dump and restarts. If you see message IOEZ00188I before zFS initialization is complete (before message IOEZ00055I), either increase the REGION size in the ZFS PROC or decrease some cache sizes in the IOEFSPRM 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.

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 sysplex-aware, 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).

Diagnosing disabled aggregates

If an internal error is detected by zFS (causing a 2C3 abend) on an aggregate that is mounted R/W, zFS will attempt to isolate the failure rather than taking zFS down. As a result, zFS might mark an aggregate unavailable and issue a message similar to IOEZ00422E.

IOEZ00422E Aggregate PLEX.JMS.AGGR001.LDS0001 disabled for writing

This is in addition to a dump and possibly zFS trace information. 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 write to the aggregate. Other aggregates that are not involved in the failure remain available. The disabled aggregate will be unavailable for writing until it is unmounted and mounted.

Note: Even though the aggregate is disabled, z/OS UNIX System Services will continue to display the aggregate mounted as R/W. To determine if the aggregate has been marked as disabled, use the `zfsadm lsaggr` command or the `zfsadm aggrinfo` command.

An aggregate that has been disabled might potentially be corrupted. (zFS has had an internal problem and has disabled the aggregate in order to avoid writing anything incorrect into the aggregate. However, because this is an internal failure, zFS cannot guarantee that the aggregate has no internal inconsistencies.) In order to be sure the aggregate is internally consistent, run the ioeagslv utility against the aggregate that was disabled. See “Understanding the salvager utility” on page 75 and “ioeagslv” on page 105 for information about running the ioeagslv utility.

Disabled compatibility mode aggregate

Compatibility mode aggregate can become disabled for a variety of reasons:

- I/O error or failure of a DASD device
- Loss of connectivity to DASD device
- Internal zFS error (zFS disables the aggregate for protection)
- Permanent corruption of aggregate.

The procedure to follow depends on the release level of zFS:

- For releases V1R11 and above that run sysplex-aware (sysplex=on in IOEFSPRM), see “Disabled aggregates when all systems are on V1R11 or later.”
- For releases before V1R11 or non-sysplex aware systems (sysplex=off in IOEFSPRM), see “Disabled aggregates when any systems are on V1R10 or earlier” on page 85.

Disabled aggregates when all systems are on V1R11 or later

The following information applies when all system are on V1R11 and above.

If the compatibility mode aggregate becomes disabled because of I/O error or corruption, unmount the file system and any other file systems mounted below the disabled aggregate. Run the ioeagslv utility to ensure that the aggregate is internally consistent. If you do not unmount before running ioeagslv, the system issues messages similar to the following example:

IKJ56225I DATA SET PLEX.JMS.AGGR001.LDS0001 ALREADY IN USE, TRY LATER+
IKJ56225I DATA SET IS ALLOCATED TO ANOTHER JOB OR USER
IOEZ00003E While opening minor device 1, could not open dataset PLEX.JMS.AGGR001.LDS0001.

After you run ioeagslv and 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 is not disabled any longer. If there are file systems mounted below the disabled aggregate, you can use the remount capability of z/OS UNIX in order 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, then you can keep the mount read-write by entering the following TSO/E UNMOUNT command:

```
UNMOUNT FILESYSTEM('PLEX.JMS.AGGR001.LDS0001') REMOUNT(SAMEMODE)
```

or the following OMVS chmount command:

```
/usr/sbin/chmount -s /zfsmnt1
```

or use the "Work with Mounted File Systems" ISHELL panel:

```
Select the attribute to change:
1. Change mount mode to R/W
2. Change Owning system from SY1
3. Change automove attribute...
4. Remount Samemode (R/O)
New owning system ________
```

For additional information about these commands, see the topics on UNMOUNT and chmount in z/OS UNIX System Services Command Reference, SA22-7802.

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.

**Disabled aggregates when any systems are on V1R10 or earlier**

The following information applies to releases before V1R11.

Unmount the compatibility mode aggregate (file system) and any other file systems mounted below the disabled aggregate. Run the ioeagslv utility to ensure that the aggregate is internally consistent. If you do not unmount before running ioeagslv, the system issues messages similar to the following example:

```
IKJ56225I DATA SET PLEX.JMS.AGGR001.LDS0001 ALREADY IN USE, TRY LATER+
IKJ56225I DATA SET IS ALLOCATED TO ANOTHER JOB OR USER
IOEZ00003E While opening minor device 1, could not open dataset
PLEX.JMS.AGGR001.LDS0001.
```

If you are in a shared file system sysplex environment (before V1R11) and one member loses DASD access, if another member still has access it can become the owner. See the topic on "Moving file systems in a sysplex" in z/OS UNIX System Services Planning, GA22-7800.

After you have run ioeagslv and 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 is not disabled any longer. If there are file systems mounted below the disabled aggregate, you can use the
remount capability of z/OS UNIX in order to avoid unmounting those lower file systems. Remount allows you to change a mounted file system from read-only to read-write or from read-write to read-only without affecting lower mounted file systems.

**Note:** If you are in a shared file system environment, you must be at z/OS V1R5 or higher to use the remount capability. Otherwise, you need to unmount the file system (and possibly lower file systems) and then mount the file system.

For example, if PLEX.JMS.AGGR001.LDS0001 is mounted read-write at /zfsmnt1 change it to a read-only mount by entering the following TSO/E UNMOUNT command:

```
UNMOUNT FILESYSTEM('PLEX.JMS.AGGR001.LDS0001') REMOUNT(READ)
```

or the following OMVS chmount command:

```
/usr/sbin/chmount -r /zfsmnt1
```

**Note:** You might see messages such as:

- When entering the TSO/E UNMOUNT command:
  
  ```
  RETURN CODE 0000008D, REASON CODE EF096271.
  THE UNMOUNT FAILED FOR FILE SYSTEM PLEX.JMS.AGGR001.LDS0001.
  ```

- When entering the OMVS chmount command:

  ```
  FOMFO504I remount error: 8D EF096271
  EROFS: The specified file system is read only
  ```

In either of these cases, the message indicates that the remount to read-only failed because zFS needs to run log recovery for this aggregate. In this case, z/OS UNIX will mount the file system back to read-write and you can ignore these messages. You can verify that the file system is mounted read-write by using the z/OS UNIX `df -v` command against the mount point (for example, `df -v /zfsmnt1`). In that case, you can skip the next step (remount or chmount to read-write).

If this succeeds, then you should change it back to read-write mode with one of the following commands:

```
UNMOUNT FILESYSTEM('PLEX.JMS.AGGR001.LDS0001') REMOUNT(RDWR)
```

or

```
/usr/sbin/chmount -w /zfsmnt1
```

You will have accomplished your unmount and mount.

If the remount (to read-only) fails, then z/OS UNIX will attempt to mount it read-write again. If this succeeds, then you have accomplished your unmount and mount. If it fails, then the aggregate will be unavailable for writing until zFS is stopped and restarted.

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.

**Disabled multi-file system aggregate**

All file systems in a multi-file system aggregate that are mounted must be unmounted before the aggregate can be detached. Of course, if there are other file systems mounted on these file systems, they must be unmounted also. After all the file systems have been unmounted, the aggregate can be detached. If this is successful, then you should then run the ioeagslv utility to ensure that the aggregate is internally consistent. If you do not detach before running ioeagslv, the system issues messages similar to the following example:

```
IKJ56225I DATA SET PLEX.JMS.AGGR001.LDS0001 ALREADY IN USE, TRY LATER+
IKJ56225I DATA SET IS ALLOCATED TO ANOTHER JOB OR USER
IOEZ00003E While opening minor device 1, could not open dataset
PLEX.JMS.AGGR001.LDS0001.
```
If the detach is unsuccessful, then the aggregate will be unavailable for writing until zFS is stopped and restarted.

To determine the names of the file systems contained within the disabled aggregate, enter the `zfsadm lsfs -aggregate` command (see "zfsadm lsfs" on page 149).

For each file system in that aggregate that is mounted, unmount it by using the TSO/E UNMOUNT command:

```
UNMOUNT FILESYSTEM(file system name)
```

or the following OMVS command:

```
/usr/sbin/unmount pathname
```

Next, detach the aggregate using the following command:

```
zfsadm detach -aggregate name
```

Then, run ioeagslv. After you have run ioeagslv and are satisfied that the aggregate is in a consistent state, attach the aggregate using the following command:

```
zfsadm attach -aggregate name
```

Then, mount the file systems that were previously mounted using the TSO/E MOUNT command:

```
MOUNT FILESYSTEM(file system name) TYPE(ZFS) MODE(RDWR) MOUNTPOINT(pathname)
```

or the following OMVS command:

```
/usr/sbin/mount -t ZFS -f filesystemname pathname
```
Chapter 10. 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 intention of the auditid is to identify a z/OS UNIX file or directory in an SMF audit record or in certain authorization failure messages (for example, RACF message ICH408). An auditid appears in Type 80 SMF records and in the output of certain z/OS UNIX APIs (for example, stat). Beginning in z/OS V1R9, zFS allows the administrator to specify whether zFS uses a more unique auditid for a zFS file or directory, or continues to use the existing, non-unique, standard auditid.

Guideline: The auditid tool can display a file path name if you know the auditid. Find the auditid tool on the z/OS UNIX tools Web site at: [www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1ty2.html](http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1ty2.html).

Do not begin to use the unique auditid capability until all sysplex members are at z/OS V1R9 with APAR OA20614 installed. Regardless of the auditid stored in the aggregate, zFS aggregates that are owned by or moved to a system running a release before z/OS V1R9 generate the standard zFS auditids. As a result, an auditid for a file can change based on which system owns the aggregate.

The unique zFS auditid format is only supported for zFS compatibility mode aggregates. The auditid for files and directories in a backup file system (a .bak clone file system) is the same as the auditids for the files and directories in the read-write file system, except that the last four bits of the CCHH of the auditid in the backup are always binary ones.

Note: Auditids for files and directories in multi-file system aggregates are not unique.

Figure 21 shows the format of the unique zFS auditid, the standard zFS auditid, and the HFS auditid.

![Figure 21. zFS auditid examples](image)

The i-node and unique identifier together 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 auditid. 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 contained in the aggregate:

- When formatting an aggregate, you can optionally specify the -newauditfid option to store a unique auditid in the aggregate. To do this, use either the IOEAGFMT batch utility, or the z/OS UNIX zfsadm format command. If you do not specify -newauditfid, the aggregate has the standard auditfid (binary zeros).

- You can optionally specify a zFS configuration option (convert_auditfid=on) in the IOEFSPRM to control whether the aggregate’s auditid is converted from an standard format auditid to a unique auditid when a zFS file system is mounted. If you specify on, zFS converts the standard auditid to the unique auditid 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.

- 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 auditid.

#### Scenarios

1. You want all your aggregates to remain with the standard auditid (and therefore all auditids have the standard format):
   a. Do nothing. The default is convert_auditfid=off in your IOEFSPRM configuration file.

   **Result:** Existing aggregates are not converted to contain unique auditfids on MOUNT (attach). Do not specify the -newauditfid option when you format a new aggregate and the standard auditid remains.

2. You want your new aggregates to have the unique auditid and your existing aggregates to remain with the standard auditid:
   a. Specify (or default to) convert_auditfid=off in your IOEFSPRM configuration file.
   b. Specify the -newauditfid option when you 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 have the unique auditid (and therefore all auditids use the new method):
   a. Use the zfsadm config -convert_auditfid on command and specify convert_auditfid=on in your IOEFSPRM configuration file. This makes auditid available on the next IPL.

   **Result:** Any existing aggregates are converted to the unique auditid the next time they are mounted (attached). When you format new aggregates and specify the -newauditfid option, the aggregates have the unique auditid.

#### Guideline:
New aggregates formatted with ISHELL, automount allocany, allocuser, or the BPXWH2Z utility do not have the unique auditfids unless you enable conversion on mount. To enable conversion, specify convert_auditfid=on in IOEFSPRM or specify -convert_auditfid on the zfsadm config command.

If a zFS aggregate is moved to another DASD location, the auditid remains the same, unless you change it using the zfsadm setauditfid -force command. This is a trade-off between changing the auditid, 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 auditfid, the standard zFS auditfid, and HFS auditid (that does not depend on the internal contents of the new zFS auditid).

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 discusses the zSeries File System (zFS) reference information.

- Chapter 11, “z/OS system commands,” on page 95
- Chapter 12, “zFS commands,” on page 101
- Chapter 13, “zFS data sets,” on page 165
- Chapter 14, “zFS application programming interfaces,” on page 177
Chapter 11. 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 1.

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 procname,query,{level | settings | threads[,allwait]}
  [[kn | vm | lfs | lock | storage | file | stkm | ctkc | svi | all]]`
- `modify procname,reset,{kn | vm | lfs | lock | storage | file | stkm | ctkc | svi | all}`
- `modify procname,trace,{reset | print}`
- `modify procname,abort`
- `modify procname,dump`
- `modify procname,hangbreak`
- `modify procname,unquiesce,aggregate_name`
- `modify procname,nsvalidate[,print]`

**Parameters**

- **procname** The name of the zFS PFS PROC. The default procname 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 XCF communication interface level (1, 2 or 3). The zFS sysplex level is controlled by the IOEFSRPRM sysplex=on configuration option. For an example and more information, see “Determining service levels” on page 78.
  - **settings** Displays the zFS configuration settings, which are based on the IOEFSRPRM file and defaults.
  - **threads[,allwait]** Displays the threads being monitored by the zFS hang detector. To display all zFS threads, use the `modify zfs,query,threads,allwait` command.
    - **Note:** The time of day values are shown in Greenwich mean time (GMT).
  - **<report>** Report consists of the following options:
    - **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.
modify zfs process

| lfs | Displays the file system statistics including the performance of the zFS metadata caches, the vnode cache and the aggregate I/O statistics. |
| lock | Displays the lock contention values. |
| storage | Displays the zFS storage values. |
| file | Displays the requests per zFS files system and aggregate. |
| stkm | Displays the current server token manager (STKM) statistics. |
| ctkc | Displays the client calls to other systems. |
| svi | Displays the calls from other systems to this server through the server vnode interface (SVI) component. |
| all | Displays all the zFS counters. |

For more details on these reports, see to "Monitoring zFS performance" on page 60.

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.
- lock
  Resets the lock contention values.
- storage
  Resets the zFS storage counters.
- file
  Resets the requests for zFS files 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.
- 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 "Performance and debugging," on page 57 for more information of the trace output data set.

abort
Causes the zFS PFS to abnormally terminate and dump. 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. The internal trace table is not printed. The modify zfs,trace,print command can be used if the internal trace table is necessary.
modify zfs process

| **hangbreak** | Causes zFS to post with a failure any requests in zFS that are waiting and are suspected of being hung by the hang detector. It will not break threads working on administration tasks. This might break and resolve the hang condition. This should only be used if you suspect that there is a hang involving zFS. The modify zfs,query,threads operator command can be used to determine if one or more requester threads remain in the same wait over several queries. If this command does not successfully break the hang, you will need to stop or cancel zFS. If you suspect that zFS is in an infinite loop, you will need to cancel zFS. For additional information, see "Steps for resolving a zFS hang" on page 80.

| **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.

| **nsvalidate** | Initiates the zFS namespace validation on the system where the command is entered. The modify nsvalidate command is typically 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 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 obtained after validation.

**Usage**
The modify zfs command is used to display zFS counters or values and to initiate or gather debugging information.

**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 dump and terminate:
modify zfs,abort

**Related Information**
File:
IOEFSPRM

For details on stopping zFS, see in the topic on Recyling z/OS UNIX System Services in z/OS MVS System Commands, SA22-7627.
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).

Format
setomvs reset=(xx)

Parameters
xx 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, SA22-7627](#).
setomvs reset
Chapter 12. zFS commands

This section provides a description of the relevant zFS commands and batch utilities. In the options section of this chapter, command options are 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 the z/OS UNIX System Services Command Reference, SA22-7802.

Note: The bpxmtext command is not valid for zFS abend reason codes (EAxxnnnn).

In z/OS V1R10 and above, you can use the man command 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. Examples of the zFS man commands are shown in Table 1.

Table 1. zFS man command examples

<table>
<thead>
<tr>
<th>zFS command</th>
<th>man command</th>
</tr>
</thead>
<tbody>
<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 zfsadmqmry</td>
</tr>
</tbody>
</table>

For more information about the man command, see

- man — Display sections of the online reference manual in z/OS UNIX System Services Command Reference, SA22-7802
- Enabling the man pages in z/OS UNIX System Services Planning, GA22-7800
ioeagfmt

ioeagfmt

Purpose
This is a batch utility that formats a VSAM Linear Data Set to become an HFS compatibility mode aggregate or a multi-file system aggregate.

Format

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 upper case. 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).

The name can be no longer than 44 characters. If this is a compatibility mode aggregate (see the -compat option), and you intend to clone the file system (see the zfsadm clone command), you might want to limit the aggregate name to 40 characters.

-compat
Indicates that a compatibility mode aggregate should be created. This means that in addition to formatting the VSAM LDS as a zFS aggregate, a zFS file system by the same name (the aggregate name) is created and its quota is set to the size of the available blocks on the aggregate. This option should normally be specified unless you want to create a multi-file system aggregate. See Chapter 8, “Multi-file system aggregates,” on page 49 for more information about multi-file system aggregates.

-group gid | name
Specifies the group owner for the root directory of the file system. This is used with the -compat option, otherwise it is ignored. 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 8K 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 specified with this option are ignored.

-initialempty blocks
Specifies the number of 8K blocks that will be left empty at the beginning of the aggregate. The default is 1. If you specify 0, you will get 1 block. This option is not normally specified.

-level
Prints the level of the ioeagfmt command. This is useful when you are diagnosing a problem. Except for -help, all other valid options specified with -level are ignored.

-logsize blocks
Specifies the size in 8K blocks of the log. The default is 1% of the aggregate size or 128
megabytes, whichever is smaller. This 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.

-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 | name

Specifies the owner for the root directory of the file system. This is used with the -compat
option, otherwise it is ignored. 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. This is used with the
-compat option, otherwise it is ignored. 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 8K 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 (LDS). 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
LDS is attempted after the primary allocation is formatted unless the -grow option is
specified. In that case, multiple extensions of the amount 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 LDS for this extension to occur. Space must be available on the
volume.

Usage

The ioeagfmt utility is used to format an existing VSAM LDS as a zFS aggregate. All zFS aggregates
must be formatted before use (including HFS compatibility mode aggregates). You can run ioeagfmt even
if the zFS PFS is not active on the system. The size of the aggregate is as many 8K blocks as fits in the
primary allocation of the VSAM LDS 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 LDS has a SHAREOPTIONS value of other than 3, ioeagfmt will change it
to SHAREOPTIONS 3 during format.

Privilege Required

The user must have ALTER authority to the VSAM LDS or must be UID 0 or have READ authority to the
SUPERUSER.FILESYS.PFSCTL profile in the z/OS UNIXPRIV class. In fact, UPDATE authority to the
VSAM LDS 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 profile in the z/OS UNIXPRIV class.

Examples

Figure 22 on page 104 shows an example of a job that creates a compatibility mode aggregate and file
system.
**ioeagfmt**

```plaintext
//USERIDA JOB,'Compatibility Mode',
//    CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//SYSDUMP 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 0) SHAREOPTIONS(3))
/*
//CREATE EXEC PGM=IOEAGFMT,REGION=0M,
// PARM=('aggregate OMVS.PRV.COMPAT.AGGR001 -compat')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*/
```

*Figure 22. Job to create a compatibility mode aggregate and file system*

**Note:** In the PARM=('aggregate OMVS.PRV.COMPAT.AGGR001 -compat') statement, the options
-aggregate and -compat must be in lower case.
ioeagslv

Purpose
This 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.

Note: 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 for compatibility mode aggregates, the next time the file system is mounted). This normally brings the aggregate (and all the file systems) back to a consistent state. The aggregate must not be mounted (or attached) when ioeagslv is run.

Format
ioeagslv -aggregate name [-recoveronly] [(converttov3 | verifyonly | salvageonly)] [-verbose] [-level] [-help]

Options
-aggregate name
Specifies the name of the aggregate to be verified, recovered, or salvaged.
-converttov3
Directs the Salvager to convert the specified aggregate. The aggregate is converted from a version 1.4 aggregate to a version 1.3 aggregate. This should normally not be necessary. It is needed if you did not install tolerance APAR OA11573 on prior releases (prior to z/OS Version 1 Release 7) before installing z/OS Version 1 Release 7. You should, of course, install OA11573 on prior releases as soon as possible. See “Usage” for information about using and combining the command’s 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 ioeagslv command. This is useful when you are diagnosing a problem. Except for -help, all other valid options specified with -level are ignored.
-recoveronly
Directs the Salvager to recover the specified aggregate. The Salvager replays the log of metadata changes that resides on the aggregate. See “Usage” 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” for information about using and combining the command’s options.
-verbose
Directs the Salvager to produce detailed information about the aggregate as it executes. The information is useful primarily for debugging purposes. It is displayed on standard output (which can be redirected). Use this option alone or with any combination of the available options.
-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” for information about using and combining the command’s options.

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 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 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 `-recoveronly`, `-verifyonly`, `-salvageonly`, and `-converttov3` options to indicate the operations the Salvager is to perform on the specified aggregate, as follows:

- **Specify the `-recoveronly` 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 `-converttov3` option**
  To convert a zFS aggregate that is in version 1.4 format to version 1.3 format. The conversion will succeed only if all file systems and the aggregate are successful converted. If the conversion is interrupted before completion, it must be run again to completion. An attempt to mount or attach an aggregate that has been partially converted will be denied.

- **Specify the `-verifyonly` option**
  To determine whether the structure of the aggregate contains any inconsistencies without running recovery or attempting to repair any inconsistencies found on the aggregate. Use this option to assess the extent of the damage to an aggregate. The Salvager makes no modifications to an aggregate during verification. Note that it is normal for the Salvager to find errors when it verifies an aggregate that has not been recovered; the presence of an unrecovered log on an aggregate makes the findings of the Salvager, positive or negative, of dubious worth.

- **Specify the `-recoveronly` and `-verifyonly` options**
  To run recovery on the aggregate and then analyze its structure without attempting to repair any inconsistencies found on it. Use these options if you believe replaying the log can return consistency to the aggregate, but you want to verify the consistency of the aggregate after recovery is run. Recovering an aggregate and then verifying its structure represents a cautious application of the Salvager.

- **Specify the `-salvageonly` option**
  To attempt to repair any inconsistencies 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 `-recoveronly`, `-verifyonly`, and `-salvageonly` options**
  To run recovery on the aggregate and then attempt to repair any inconsistencies 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 found on the aggregate. (Omitting the three options is equivalent to specifying the `-recoveronly` and `-salvageonly` options.)

In some cases, when repairing an aggregate, it might be necessary to attempt the repair several times before the repair is complete. 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 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 z/OS 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 contained in files on the aggregate. If it makes changes,
the Salvager displays the path names of the files affected by the modifications, when the path names can be determined. The owners of the files can then verify the files’ contents, and the files can be restored from backups if necessary.

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 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 systems 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.

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 found in earlier passes to help in the analysis.

In general, the Salvager exits with an error code of at least 16 without analyzing a VSAM LDS that it is sure is not a zFS aggregate. It also exits with an error code of 16 if a file system on the aggregate to be recovered or salvaged is attached. (If necessary, you can use the zfsadm detach command to detach the aggregate.)

As the Salvager executes, it maintains a number of internal lists. Each list consists of anodes that failed verification in specific ways. When it initially scans an aggregate, the Salvager marks as “unsafe” anodes with which it encounters problems. The Salvager later attempts to determine the actual path names associated with these anodes to include the path names in the lists. When it has finished salvaging, the Salvager displays any non-empty lists.

For addition details about running the salvage utility, see “Understanding the salvager utility” on page 75.

Privilege Required
The user needs UPDATE authority for the specified VSAM LDS or the user must be uid 0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the z/OS UNIXPRIV class. 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 profile in the z/OS UNIXPRIV class.

Examples
Figure 23 on page 108 shows an example of a job that invokes the ioeagslv utility.
ioeagslv

//USERIDA JOB ,'Salvage',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//SALVAGE EXEC PGM=IOEAGSLV,REGION=0M,
// PARM=('aggregate OMVS.PRIV.COMPAT.AGGR001 -verifyonly')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*

Figure 23. Job to verify a zFS aggregate
MOUNT

Purpose
This 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, SA22-7802.

Note: An attempt to mount a zFS file system that is contained in a zFS multi-file system aggregate running in a sysplex will be denied.

Format
MOUNT TYPE(file_system_type) [PARM(parameter_string)]

Options

TYPE (file_system_type)
Specifies the file system type. In z/OS V1R7 and above, to aid migration from HFS, the TYPE option is generic. 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 you specify the wrong file type (for example, HFS instead of zFS), any associated parameter string is ignored. For additional information, see Mounting considerations in z/OS UNIX System Services Planning, GA22-7800.

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.

PARMs for compatibility mode aggregates actually apply to the attach of the aggregate. The attach occurs on the first (or only) mount issued for the aggregate. The PARMs are ignored on a mount for an aggregate that is already attached. For example, if a read-write file system in a compatibility mode aggregate is mounted and then the backup file system (the .bak) is mounted, the attach occurs on the mount of the read-write file system. The PARMs are ignored on the mount of the backup file system because the aggregate is already attached. However, you must still specify the PARMs on both mounts in case the order in which the mounts will occur is not clear. The PARMs must be the same on each mount (except that you might consider the RW PARM on the mount of the backup file system if you want to ensure that the aggregate is attached R/W in case the backup file system is mounted first).

The following parameters apply to both types of aggregates (compatibility mode aggregates and multi-file system aggregates):

FSFULL(threshold,increment)
Specifies the threshold and increment for reporting file system quota error messages to the operator. The default is the fsfull specification in the IOEFSRPM file.

READAHEAD | NOREADAHEAD
Specifies whether this file system will be accessed sequentially or not and whether the zFS read ahead processing normally done should be enabled or disabled. NOREADAHEAD should be not be specified unless under the direction of IBM Service. The default is to do read ahead processing.

The following parameters apply to compatibility mode aggregates:
Note: These options are only effective when the mount causes an attach, which is normally the case. However, if the backup file system is mounted first, that is when the attach is done. A subsequent mount of the read-write file system does not cause an attach and any options specified (because these are all aggregate options) have no effect.

AGGFRFULL(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. This parameter only applies to compatibility mode aggregates/file systems.

AGGRGROW | NOAGGRGROW
Specifies whether the aggregate is eligible to be dynamically grown. The growth will be 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. This parameter only applies to compatibility mode aggregates/file systems.

NBS | NONBS
Specifies the new block security processing for this aggregate. The default is the nbs specification in the IOEFSprm file. This parameter only applies to compatibility mode aggregates/file systems.

RW
Specifies that the aggregate is to be attached R/W even though the file system is being mounted R/O. The default is to attach the aggregate R/O when the file system is mounted R/O. It is typically used when the backup file system (.bak) is mounted before the read-write file system is mounted. This parameter only applies to compatibility mode aggregates/file systems.

The following parameters apply to multi-file system aggregates:

AGGREGATE(aggregate_name)
Specifies the name of the aggregate that the file system resides in. This is normally used when the zFS file system you are mounting has the same name as a zFS file system in another aggregate. See also the FILESYSTEM parameter. This parameter only applies to multi-file system aggregates/file systems.

FILESYSTEM(zFS_filesystem_name)
Specifies the name of the zFS file system that you are mounting. This is normally used when the zFS file system you are mounting has the same name as a zFS file system in another aggregate. If this is not specified, zFS assumes that the zFS file system name is the same as the z/OS UNIX file system name (specified in the MOUNT FILESYSTEM option). This parameter only applies to multi-file system aggregates/file systems. The FILESYSTEM parameter cannot be specified without the AGGREGATE parameter.

FSGROW(increment,times)
Specifies that the file system quota is to be dynamically grown when the file system becomes full (that is, reaches its quota). The increment specifies how much the quota is to grow in K-bytes. The times specifies how many times the quota is to be grown before the file request is denied. The default is the fsgrow specification in the IOEFSprm file. The maximum value that can be specified is 2147483647. If the physical space becomes exhausted, the aggrgrow specification in the IOEFSprm file controls whether the aggregate is dynamically grown. This parameter only applies to multi-file system aggregates/file systems. It is not saved across attaches.

Usage
The MOUNT command mounts a zFS file system.
**MOUNT** of a compatibility mode aggregate is serialized with other **zfsadm commands** (because **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 indicates that the zFS aggregate indicator can not 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 for Catalogs, SC26-7394](https://www.ibm.com/support/pages/zos-documentation-access-meth)). 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)')
```

Here is the same example as an OMVS command:

```
/usr/sbin/mount -f OMVS.PRV.AGGR004.LDS0004 -t ZFS -o 'AGGRFULL(90,5)' /etc/zfscompat1
```

The following TSO/E example mounts a zFS file system and specifies a z/OS UNIX file system name that is different from the zFS file system name (because another zFS file system with the same name (in a different aggregate) has already been mounted):

```
MOUNT FILESYSTEM('OMVS.PRV.FS1.DUP1') MOUNTPOINT('/etc/zfsmntpt2') TYPE(ZFS) MODE(RDWR)
   PARM('AGGREGATE(OMVS.PRV.AGGR005.LDS0005),FILESYSTEM(OMVS.PRV.FS1)')
```

Here is the same example as an OMVS command:

```
/usr/sbin/mount -f OMVS.PRV.FS1.DUP1 -t ZFS -o AGGREGATE(OMVS.PRV.AGGR005.LDS0005),
   FILESYSTEM(OMVS.PRV.FS1) /etc/zfsmntpt2
```

The following TSO/E example mounts a clone of a zFS file system by specifying triple apostrophes to preserve the mixed case file system name.

```
MOUNT FILESYSTEM('''OMVS.PRV.FS3.bak''') TYPE(ZFS) MODE(READ) MOUNTPOINT('/etc/zfsmntpt3')
```

Here is the same example as an OMVS command:

```
/usr/sbin/mount -f OMVS.PRV.FS3.bak -t ZFS -r /etc/zfsmntpt3
```

**Related Information**

Command:
MOUNT

UNMOUNT (For information about this command, see z/OS UNIX System Services Command Reference, SA22-7802)

File:
  IOEFSPRM
**zfsadm**

**Purpose**
Introduction to the *zfsadm* command suite. The *zfsadm* command is executed 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 24 on page 119](#) 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 bold 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 executing 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 documentation. 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 in alphabetic order to make them easier to locate—this does not reflect the format of the command. The formats are presented that same as on your system.

- **Arguments** - Arguments for options always appear in italic type 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.

- **Optional information** - 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 115.

- `-size kbytes`
  Specifies the size in K-bytes for the kbytes argument.

- `-system system name`
  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. They apply to multi-file system aggregates although several apply to compatibility mode aggregates too (for example, `zfsadm grow` and `zfsadm quiesce/unquiesce`). 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 24 on page 119. The descriptions of the `zfsadm aggrinfo` and the `zfsadm attach` commands show examples 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.

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 three `zfsadm` commands that begin with `l`: `zfsadm lsaggr`, `zfsadm lsfs`, and `zfsadm lsquota`. To remain unambiguous, they can be abbreviated to `zfsadm lsa`, `zfsadm lsf`, and `zfsadm lsq`.

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
```

**Note:** 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 cancelling 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`.

**Privilege Required**

`zfsadm` commands that query information (for example, `lsfs`, `aggrinfo`) can be issued by any user that has READ authority to the data set that contains the `IOEFSPRM` file. `zfsadm` commands that modify (for example, `setquota`, `create`) additionally require that the issuer must be one of the following:

- UID of 0

  **Note:** 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.

- Have READ authority to the SUPERUSER.FILESYS.PFSCTL profile 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 clone`
- `zfsadm clonesys`
- `zfsadm config`
- `zfsadm configquery`
- `zfsadm create`
- `zfsadm define`
- `zfsadm delete`
zfsadm

zfsadm detach
zfsadm format
zfsadm grow
zfsadm help
zfsadm lsaggr
zfsadm lsfs
zfsadm lsquota
zfsadm lssys
zfsadm query
zfsadm quiesce
zfsadm rename
zfsadm setauditfid
zfsadm setquota
zfsadm unquiesce

Files:

IOEFSPRM
zfsadm aggrinfo

Purpose
Displays information about an aggregate, or all attached aggregates, if there is no specific aggregate specified.

Format
zfsadm aggrinfo [-aggregate name | -system system name] [-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 upper case. 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 output of the command to be shortened to display only the aggregate name if it contains one or more file systems or a message indicating that there are no file systems contained in the aggregate.

-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.

-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)
- number of free 8K blocks
- number of free 1K fragments
- size of the log file
- size of the filesystem table
- size of the bitmap file.

-system system name
Specifies the name of the system the report request will be sent to, to retrieve the data requested.

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.
zfsadm aggrinfo

- Whether the aggregate is read-write (R/W) or read-only (R/O), it is a mounted compatibility mode
  aggregate (COMP), a multi-file system aggregate or an attached compatibility mode aggregate (MULT),
  or the aggregate is currently quiesced (QUIESCED), disabled (DISABLED), or both.
- The amount of space available in K-bytes.
- The total amount of space in the aggregate in K-bytes. (To grow an aggregate using the zfsadm
  command, specify a number larger than this number.)
- If -long is specified, the version of the aggregate, the auditid, the number of free 8K blocks, the
  number of free 1K fragments, the size of the log file, the size of the file system table and the size of
  the bitmap file.

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 about the disk space available on all aggregates attached on
the system:

zfsadm aggrinfo -long
IOEZ00369I A total of 5 aggregates are attached to the sysplex.  
PLEX.JMS.AGGR004.LDS0004 (R/W COMP): 500 K free out of total 12960
  version 1.4
  auditid C3C6C3F0 F0F0051E 0000
  55 free 8k blocks; 60 free 1K fragments
  112 K log file; 24 K filesystem table
  8 K bitmap file
PLEX.JMS.AGGR006.LDS0006 (R/W COMP): 71206 K free out of total 72000
  version 1.4
  auditid 00000000 00000000 0000
  8899 free 8k blocks; 14 free 1K fragments
  720 K log file; 40 K filesystem table
  16 K bitmap file
PLEX.JMS.AGGR005.LDS0005 (R/W COMP): 1276901 K free out of total 1302480
  version 1.4
  auditid C3C6C3F0 F0F10000 0000
  159610 free 8k blocks; 21 free 1K fragments
  13032 K log file; 56 K filesystem table
  192 K bitmap file
PLEX.JMS.AGGR008.LARGE08 (R/W COMP): 1426971 K free out of total 1872720
  version 1.4
  auditid C3C6C3F0 F0F006A0 0000
  178368 free 8k blocks; 27 free 1K fragments
  18736 K log file; 56 K filesystem table
  272 K bitmap file
PLEX.JMS.AGGR002.LDS0002 (R/O MULT): 4886 K free out of total 5040
  version 1.3
  auditid 00000000 00000000 0000
  609 free 8k blocks; 14 free 1K fragments
  112 K log file; 24 K filesystem table
  8 K bitmap file
Figure 24 shows the same example as a job that invokes **zfsadm aggrinfo**.

```plaintext
//USERIDA JOB 'Zfsadm Aggrinfo',
  //     CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
  //AGGRINFO EXEC PGM=IOEZADM,REGION=0M,
  // PARM=('aggrinfo -long')
  //SYSPRINT DD SYSOUT=H
  //STDOUT DD SYSOUT=H
  //STDERR DD SYSOUT=H
  //SYSUDUMP DD SYSOUT=H
  //CEEDUMP DD SYSOUT=H

Figure 24. Job to display aggregate information

**Related Information**

**Command:**

```
zsadm lsaggr
```

**File:**

```
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.

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
lsquota: list filesystem and aggregate space usage

Related Information
Command:
  zfsadm help
zfsadm attach

Purpose
Attaches an aggregate to zFS (without mounting the file system). In z/OS V1R11 and above, you cannot attach a zFS multi-file system aggregate in a sysplex shared file system environment. You can attach a zFS compatibility mode aggregate (a zFS aggregate that contains one read-write file system and possibly a backup (.bak) file system) in any environment.

Note: zfsadm aggrinfo displays an attached compatibility mode aggregate as MULT because it is not mounted.

Format
zfsadm attach {-aggregate name [-system system name] | -all} [-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 upper case. This aggregate does not need an entry in the IOEFSPRM file. If the aggregate is not contained in the IOEFSPRM file, it needs to be attached again if it is a multi-file system aggregate and the zFS PFS is restarted.

Note: Compatibility mode aggregates do not need to be attached with the zfsadm attach command, nor do they need to be contained in the IOEFSPRM file. Compatibility mode aggregates 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. This overrides the aggrfull option in the define_aggr entry for this aggregate in the IOEFSPRM file and the global aggrfull entry in the IOEFSPRM. 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 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.

-all
Specifies that all aggregates listed in the IOEFSPRM file available to this system that are not currently attached are to be attached.

-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.

-nbs
Specifies whether New Block Security is used for file systems in this aggregate. New block security refers to the guarantee made when a system fails. If -nbs is specified, then we guarantee that at the time of a failure. If a file was being extended or new blocks were being allocated for the file, but the user data has not yet made it to the disk when the failure occurred, then we show the newly allocated blocks as all binary 0’s and not whatever was on disk in those blocks at time of failure. The default for this is the global nbs entry in the IOEFSPRM file.
zfsadm attach

-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.

-nonbs Specifies that the New Block Security guarantee is not required. See the explanation of -nbs for a description of the New Block Security guarantee.

-R/O | -ro Specifies that the aggregate should be opened in read-only mode. A read-only aggregate means that all file systems are read-only and can only be mounted as read-only. 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  system name 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 upper case.

Usage

The zfsadm attach command attaches zFS aggregates on this system.

If the -all option is provided, the command attaches all aggregates listed in the IOEFSPRM file that is available to this system. If the -aggregate option is provided, only the aggregate specified is attached. The specified name need not be listed in the IOEFSPRM file.

When zfsadm attach -all executes, it reads the IOEFSPRM file that is available to this system to determine the aggregates to be attached. All aggregates will be attached. If an aggregate is already attached, this will be indicated. If the attach fails because log recovery is unsuccessful, you can run the ioeagslv command with the -verifyonly option on the aggregate to determine if there is an inconsistency. If this is the case, use the ioeagslv command to recover the aggregate that caused the failure and reissue the zfsadm attach command.

The zfsadm lsaggr command can be used to display a current list of all aggregates attached on this sysplex with the zFS owning system indicated, or this system when -system is used.

If the DASD volume containing the zFS multi-file system aggregate being attached is read-only, you might receive message IOEZ00336I. This indicates that the zFS aggregate indicator can not 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.

For multi-file system aggregates, define_aggr entries are generally included in the IOEFSPRM file for them rather than issuing zfsadm attach commands at the keyboard. After included in the IOEFSPRM file, all aggregates listed in the IOEFSPRM file are attached whenever zFS is started (or restarted) and auto_attach=on in the IOEFSPRM file.

Compatibility mode aggregates do not need to be separately attached because they are attached during MOUNT processing. Therefore, compatibility mode aggregates do not need define_aggr entries in IOEFSPRM, nor do they need to be attached with the zfsadm attach command.

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 profile in the z/OS UNIXPRIV class. If you are not using IOEFSPRM but instead, you are using parmlib (IOEPRMxx), the
The issuer is required to be logged in as root or to have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the z/OS UNIXPRIV class.

**Examples**

The following command attaches all of the aggregates that have entries in the system’s IOEFSPRM file.

```
zfsadm attach -all
```

The following command attaches an aggregate. No entry is needed in the system’s IOEFSPRM file.

```
zfsadm attach -aggregate OMVS.PRIV.AGGR001.LDS0001
```

Figure 25 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.PRIV.AGGR001.LDS0001")
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//CEEDUMP DD SYSOUT=H

/*

Figure 25. Job to attach an aggregate

**Note:** 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.PRIV.AGGR001.LDS0001 -R/O")

```

zFS, by default, attaches aggregates listed in the IOEFSPRM file at start-up (or restart). This is based on the auto_attach option (default is on) of the IOEFSPRM file. The `zfsadm attach` command is used if you had created and formatted a multi-file system aggregate after starting zFS and you did not want to restart zFS. A define_aggr entry for this multi-file system aggregate can be placed in the IOEFSPRM file so that it is attached the next time zFS is started.

**Related Information**

Commands:

- `zfsadm create`
- `zfsadm lsaggr`

File:

- IOEFSPRM
zfsadm clone

Purpose
Allows the administrator to make a read-only clone of a file system in the same data set. This clone file system can be made available to users to provide a read-only point-in-time copy of a file system. Only the metadata is copied (pointers to the data blocks), not the actual data. It can be used to mount a point-in-time copy of the original file system. It is not a substitute for an offline copy of the file system for media failure recovery.

Format
zfsadm clone {-filesystem name | -mfilesystem mount_name} [-aggregate name] [-level] [-help]

Options
-aggregate name
Specifies the name of the aggregate where the zFS file system name 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 upper case.

(filesystem name)
Specifies the file system name of the read-write source file system.

-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.

-mfilesystem mount_name
Specifies the z/OS UNIX file system name of the file system that is to be cloned. The file system name is case sensitive. If it was specified in upper case when the file system was mounted, it must be specified in upper case here.

Usage
This command creates a backup version (clone) of the metadata (schema) for the indicated read-write zFS file system. It names the new backup version by adding a .bak extension to the name of its read-write source file system. It places the backup version on the same aggregate as the read-write version. After the clone operation, the backup file system can be mounted read-only.

Rules: The following rules apply to the zFS clone operation:
- The aggregate that the read-write file system is contained in must be attached.
- The read-write file system can be mounted when the clone operation is issued, or not.
- You cannot clone if the backup file system is mounted.
- You cannot mount the backup file system while it is being re-cloned or deleted.
- File/directory operations against a mounted read-write file system are suspended during the clone operation.
- The zfsadm clone command cannot clone non-zFS file systems.
- If a backup version already exists, the new clone replaces it.
- If the read-write file system name is longer than 40 characters, the clone fails.
- If the clone operation takes longer than approximately 30 seconds, message IOEZ00588E is displayed on the operator console. It will be deleted (DOMed) when there are no clone operations in progress.
You can determine if a clone operation is in progress on an aggregate by entering the zfsadm lsfs -long command, which shows the clone in progress for the backup file system.
Note: The clone operation is not a substitute for a prudently scheduled backup operation. For details, see Chapter 6, “Performing a back up of zFS,” on page 43.

Privilege Required

The issuer must have READ authority to the data set that contains the IOEFSPRM file and must be root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL profile 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 profile in the z/OS UNIXPRIV class.

Examples

The following command creates a backup version of the file system OMVS.PRV.FS1:

```
zsadm clone OMVS.PRV.FS1
```

IOEZ00225I File system OMVS.PRV.FS1 successfully cloned.

Related Information

Command:
- `zsadm clonesys`
- `zsadm lsfs`
- `zsadm delete`
**zfsadm clonesys**

**Purpose**
Creates backup versions of all indicated file systems.

**Format**

```
zfsadm clonesys [-prefix string] [-aggregate name | -system system name] [-level] [-help]
```

**Options**

- **-aggregate name**
  Specifies the aggregate name of the aggregate where the read-write source file systems are stored. Omit all options to back up all file systems on the system. The aggregate name is not case sensitive. It is translated to upper case.

- **-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.

- **-prefix string**
  Specifies a character string of any length. Every file system with a name matching this string is cloned. Include field separators (such as periods) if appropriate. This option can be combined with -aggregate. Omit all options to back up all file systems on the system or -system. The prefix name is case sensitive.

- **-system system name**
  Specifies the name of the system that will be used to subset the zFS aggregates (they are zFS owned by this system) containing zFS read-write file systems to be cloned.

**Usage**
The zfsadm clonesys command creates a backup version, or clone, of each indicated read-write zFS file system. The file systems must be in aggregates that are attached. The read-write file systems can be mounted when the clonesys operation is issued, or not. The backup file systems cannot be mounted when the clonesys operation is issued. The command names each backup version by adding a .bak extension to the name of its read-write source file system. It places each backup version in the same aggregate as its read-write version. The zfsadm clonesys command cannot backup non-zFS file systems.

If a backup version of a file system already exists, the new clone replaces it.

By combining the -prefix and -aggregate options, you can create backup copies of different subsets of read-write file systems. To back up:

- All file systems in a sysplex, specify no options
- All file systems in a sysplex with a name beginning with the same character string (for example, sys. or user.), specify the string with the -prefix option
- File systems on a specific aggregate, specify the -aggregate option
- File systems with a certain prefix on a specific aggregate, specify the -prefix and -aggregate options
- File systems on a specific system, specify the -system option
- File systems with a certain prefix on a specific system, specify the -prefix and -system options

Use the zfsadm clone command to back up a single read-write zFS file system.
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.FILESYS.PFSCTL profile 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 profile in the z/OS UNIXPRIV class.

Examples
The following example creates a backup version of each zFS file system on the DCEIMGVQ system that begins with THR:
```
zfsadm clonesys -p THR -system dceimgvq
```

IOEZ00368I A total of 1 aggregates are attached to system DCEIMGVQ.
IOEZ00219I Clonesys starting for aggregate PLEX.JMS.AGGR003.LDS0003, prefix THR
IOEZ00225I File system THREE successfully cloned.
IOEZ00216I Clone ending for aggregate PLEX.JMS.AGGR003.LDS0003 (Total: 1, Failed: 0, Time 0.833)

Related Information
Command:
```
zfsadm clone
```

File:
```
IOEFSPRM
```
**zfsadm config**

**Purpose**
Changes the value of zFS configuration (IOEFSPRM) options in memory. See [Chapter 13, “zFS data sets,” on page 165](#) 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 PDSE_dataset_name] [-tran_cache_size number]
[-msg_output_dsn Seq_dataset_name] [-user_cache_readahead {on|off}]
[-metaback_cache_size number[,fixed]] [-fsgrow increment,times]
[-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]]
[-token_cache_size cache size] [-system system name]
[-level] [-help]
```

**Options**
- **Guideline:** When you change options that apply to zFS aggregates and file systems, the current default changes, but does not affect already mounted file systems until they are unmounted and remounted. This includes:
  - aggrfull
  - aggrgrow
  - fsfull
  - fsgrow
  - nbs

- **-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.

- **-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.

- **-convert_auditfid on | off**
  Specifies whether the zFS auditfid is automatically changed to the unique format on mount (attach). If on is specified, mount (attach) changes the standard auditfid format to the unique auditfid format if the mount (attach) is read-write. If off is specified or defaulted (or the mount (attach) is read-only), the auditfid is unaffected.

- **-file_threads number**
  Specifies the current number of file threads. This is only meaningful when zFS is running sysplex-aware.
-fsfull  threshold,increment
Specifies the threshold and increment for reporting file system quota full error messages to
the operator.

-fsgrow  increment,times
Specifies the increment in k-bytes and the number of times that a file system’s quota
should be increased when it becomes full.

-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 whether new block security is globally off or on by default.

-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 syncs.

-system  system name
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. 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.
zfsadm config

-user_cache_readahead on | off
  Specifies whether zFS should attempt to read ahead or not.

-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:
  • -auto_attach
  • -dir_cache_size
  • -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 IOEFSPRM file and must be root or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile 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 profile in the z/OS UNIXPRIV class.

Examples
The following example changes the size of the user cache:

cmd
EOZ03000I Successfully set -user_cache_size to 64M

Related Information
Command:
  zfsadm configquery

File:
  IOEFSPRM
zfsadm configquery

Purpose
Queries the current value of zFS configuration options.

Format
```
zfsadm configquery [-system system name] [-adm_threads] [-aggrfull] [-aggrgrow]
[-all] [-auto_attach] [-client_cache_size] [-client_reply_storage]
[-group] [-log_cache_size] [-meta_cache_size] [-metaback_cache_size]
[-syslevel] [-sysplex_state] [-token_cache_size] [-trace_dsn] [-trace_table_size]
[-tran_cache_size] [-user_cache_readahead] [-user_cache_size] [-vnode_cache_size]
[-level] [-help]
```

Options
- `adm_threads` Displays the number of threads 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.
- `auto_attach` Displays whether aggregates defined and listed in the IOEFSPRM file are attached when zFS is started.
- `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.
- `convert_auditfid` Displays whether the zFS auditfid is automatically changed to the unique format on mount (attach). If `on` is specified 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 defaulted 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.
- `fsfull` Displays the threshold and increment for reporting file system quota full error messages to the operator.
- `fsgrow` Displays the increment in k-bytes and the number of times that a file system’s quota should be increased when it becomes full.
- `group` Displays the XCF group used by zFS for communication between sysplex members.
- `help` Prints the online help for this command. All other valid options specified with this option are ignored.
zfsadm configquery

- **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 used to contain buffers for log file pages.

- **meta_cache_size**
  Displays the size, in bytes, of the cache used to contain meta data.

- **metaback_cache_size**
  Displays the size of the backing cache for meta data.

- **msg_input_dsn**
  Displays the name of the data set that contains translated zFS messages.

- **msg_output_dsn**
  Displays the name of a data set that contains any output messages that come from the zFS PFS.

- **nbs**
  Displays whether new block security is globally off or on by default.

- **romount_recovery**
  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 unmounts and mounts the aggregate again in read-only format.

- **sync_interval**
  Displays the number of seconds in the interval that zFS flushes data in the buffers to disk.

- **syslevel**
  Displays the zFS kernel (the PFS) information. It includes
  - 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 for read-write (referred to as file) or not (referred to as admin-only), and the zFS XCF protocol level (normally 3 for zFS V1R11).
    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.

- **sysplex_state**
  Displays the sysplex state of zFS.
  - Zero (0) indicates that zFS is not in a shared file system environment (normal for V1R6 and prior releases and for single system configurations including monoplex and xcflocal).
  - One (1) indicates that zFS is in a shared file system environment (normal for V1R7 and above in a shared file system environment).
  - Two (2) indicates that zFS is running in a sysplex-aware environment.

- **system system name**
  Specifies the name of the system the report request will be sent to, to retrieve the data requested.

- **token_cache_size**
  Displays the current token_cache_size maximum. Current usage is displayed through the **MODIFY ZFS,QUERY,STKM** command. This is only meaningful when zFS is running sysplex-aware.

- **trace_dsn**
  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.
-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_readahead
  Displays whether zFS should attempt to read ahead or not.

-user_cache_size
  Displays the size, in bytes, of the cache 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 may 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.

If you want to display all the zFS configuration options from each member, you can use something such as:

zfsadm lssys | grep -v IOEZ00361I | xargs -n 1 zfsadm configquery -all -system

Related Information
Command:
  zfsadm config

File:
  IOEFSPRM
**zfsadm create**

**Purpose**
Creates a read-write zFS file system in an aggregate. This is for multi-file system aggregates only.

**Format**
```
zfsadm create -filesystem name -aggregate name -size kbytes
[-owner {name | uid}]
[-group { name | gid}]
[-perms permbits] [-level] [-help]
```

**Options**
- **-aggregate name**
  Specifies the name of the aggregate where the read-write file system is to be stored. The aggregate name is not case sensitive. It is translated to upper case.

- **-filesystem name**
  Specifies a name for the read/write file system. The file system name is case sensitive. That is, if you specify the file system name in lower case on the `zfsadm create` command, you must specify the file system name in lower case when you MOUNT it. The TSO/E MOUNT command translates the file system name to upper case even if it is within quotation marks. You can avoid this translation to upper case if you specify the file system name on the TSO/E MOUNT command within triple quotation marks. For example, if you specify FILESYSTEM("lower.case.example"), the file system name is not translated to upper case. However, you might find it simpler to specify the file system name in upper case on the `zfsadm create` command. The name must be unique within the system, and it should indicate the file system’s contents. The following characters can be included in the name of a file system:
  - 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).

  The name can be no longer than 44 characters. This includes the .bak extension, which is added automatically when a backup version of the file system is created (for example, by using `zfsadm clone`). If you intend to clone this file system, you might want to limit the file system name to 40 characters. Note that the .bak extension is reserved for use with backup file systems so you cannot specify a file system name that ends with this extension.

  If you are using both multi-file system aggregates and compatibility mode aggregates, do not name any file systems in multi-file system aggregates with the same name as any of your compatibility mode aggregates. If you do this, you will get a different file system mounted depending on whether an aggregate is attached or not. For example, suppose you have compatibility mode aggregate A.B.C and you have multi-file system aggregate D.E.F that contains file system A.B.C. When you mount file system A.B.C, you will get the one in aggregate D.E.F mounted if D.E.F is attached. If D.E.F is not attached, you will get compatibility mode aggregate A.B.C mounted.

  However, you can create file systems with the same name in different multi-file system aggregates. The `zfsadm` commands and the MOUNT command can specify a zFS file system name that is qualified by its aggregate name.
-group name | gid
Specifies the group of the root directory of the file system. This can be specified as a z/OS
group ID or a numeric gid. The default is the group of the issuer of the command. If only
-owner is specified, the group is the owner’s default group.

-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.

-owner name | uid
Specifies the owner of the root directory of the file system. This can be specified as a
z/OS user ID or as a numeric uid. The default is the uid of the issuer of the command.

-perms permbits
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 kbytes
Specifies the initial maximum quota for the file system in K-bytes. The minimum value is
128 (for 128 K bytes).

Usage
The zfsadm create command creates a read-write zFS file system, names it as specified by
filesystem, and places it in the multi-file system aggregate specified by -aggregate. The aggregate must be attached.
(This is accomplished by issuing the zfsadm attach command or by placing a define_aggr entry for the
aggregate in the IOEFSPRM file and starting (or restarting) zFS.)

If this command succeeds, the file system can be made available for use by MOUNTing it into the z/OS
UNIX hierarchy. The command creates an empty root directory in the file system, which becomes visible
when the file system is mounted.

Note: You cannot create another file system in a mounted compatibility mode aggregate. If you really
want to do this, you must unmount it, attach the aggregate and then create the file system. This
will, however, change the compatibility mode aggregate into a multi-file system aggregate.

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.FILESYS.PFSCTL profile 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.FILESYS.PFSCTL profile in the z/OS UNIXPRIV class.

Examples
The following command creates the read-write file system OMVS.USER.PAT, with an initial quota of 5000
1K blocks in aggregate OMVS.PRV.AGGR001.LDS0001.

zfsadm create OMVS.USER.PAT omvs.prv.aggr001.lds0001 5000

IOEZ00099I File system OMVS.USER.PAT created successfully

Related Information
Commands:
    zfsadm delete
    zfsadm lsf

File:
zfsadm create

IOEFSPRM
zfsadm define

Purpose
Defines a VSAM Linear Data Set (VSAM LDS) in preparation to 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 system name] [-model model [catalog]] [-volumes volume [volume ...]] [-cylinders primary [secondary]] [-kilobytes primary [secondary]] [-megabytes primary [secondary]] [-tracks 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 LDS that is defined. The aggregate name is not case sensitive. It is translated to upper case.
-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 LDS 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 21" or "Dynamically growing a multi-file system aggregate" on page 53 for additional information.
-dataclass SMS_data_class  Specifies the name of the data class to be used when the VSAM LDS 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 LDS 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 21" or "Dynamically growing a multi-file system aggregate" on page 53 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 LDS is defined.
-megabytes primary [secondary]  Specifies the primary and optionally, the secondary allocation size for the VSAM LDS 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 21" or "Dynamically growing a multi-file system aggregate" on page 53 for additional information.
zfsadm define

-model model [catalog]
Specifies the name of the model and optionally, the model entry’s catalog to be used when
the VSAM LDS is defined.

-records primary [secondary]
Specifies the primary and optionally, the secondary allocation size for the VSAM LDS 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 21 or
“Dynamically growing a multi-file system aggregate” on page 53 for additional information.

-storageclass SMS_storage_class
Specifies the name of the storage class to be used when the VSAM LDS is defined.

-system system name
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 LDS 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 21 or “Dynamically growing a multi-file system aggregate” on page 53 for additional
information.

-volumes volume
Specifies the volume on which the VSAM LDS can have space.

Usage
The zfsadm define command defines a VSAM LDS. The VSAM LDS is available to be formatted as a
zFS aggregate. The command creates a DEFINE CLUSTER command string for a VSAM LDS 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 LDS.

Examples
The following command defines a VSAM LDS.

zfsadm define -aggregate omvs.prv.aggr001.lds0001 -volumes prv000 prv001 -cylinders 10 5

Related Information
Commands:
   zfsadm format
**Purpose**
Removes a file system. This is for deleting a backup file system in a compatibility mode aggregate (sometimes referred to as unclone) or for deleting file systems in a multi-file system aggregate.

**Format**
```
zfsadm delete -filesystem name [-aggregate name] [-level] [-help]
```

**Options**
- `-aggregate name`
  Specifies the name of the aggregate where the zFS file system name 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 upper case.

- `-filesystem name`
  Specifies the name of the read-write or backup file system to be removed. Include the `.bak` extension if specifying the name of a backup file system. The file system name is case sensitive.

- `-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.

**Usage**
The `zfsadm delete` command removes the read-write or backup zFS file system indicated by the `-filesystem` option from its aggregate. The aggregate containing the file system to be deleted must be attached. Read-write file systems and backup file systems are related during removal as follows:

- Removing a read-write file system automatically removes its associated backup version (if the backup version exists).
- Removing a backup file system does not remove the read-write file system.

File/directory operations against a mounted read-write file system are suspended during the delete of the backup file system. If the delete of a backup operation takes longer than approximately 30 seconds, message IOEZ00588E is displayed on the operator console. The message is deleted (DOMed) when there are no clone delete operations in progress. You can determine if a delete of a clone operation is in progress on an aggregate by using the `zfsadm lsfs -long` command, which shows the delete in progress for the backup file system.

If the zFS file system to be removed is also mounted, you must unmount it before you delete it. The `zfsadm delete` command cannot be used to delete a file system that is mounted. You can delete a compatibility mode file system (and its aggregate) by using the IDCAMS DELETE operation. This 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 23](#).

**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.FILESYS.PFSCTL` profile 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.FILESYS.PFSCTL` profile in the `z/OS UNIXPRIV` class.
Examples
The following command deletes the read-write file system named **OMVS.USER.PAT** and its backup version (if it exists) from its aggregate:

```
zfsadm delete OMVS.USER.PAT
```

IOEZ00105I File system OMVS.USER.PAT deleted successfully

Related Information
 Commands:
```
zfsadm clone  
zfsadm create  
zfsadm lsfs
```

File:
```
IOEFSPRM
```
**zfsadm detach**

**Purpose**
Detaches one or more aggregates from zFS. This makes any file systems contained in the aggregate unavailable to zFS. This is for multi-file system aggregates only.

**Format**
```
zfsadm detach [([-aggregate aggregate name] | -all [-system system name]]) [-level] [-help]
```

**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 upper case.
- **-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 is useful when you are diagnosing a problem. Except for -help, all other valid options specified with -level are ignored.
- **-system system name**
  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.

Before detaching an aggregate, all file systems in the aggregate must be unmounted. Therefore, **zfsadm detach -all** will not detach compatibility mode aggregates. The -system option cannot be specified without the -all option.

**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 profile 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 profile in the z/OS UNIXPRIV class.

**Examples**
The following is an example of a **zfsadm detach** command that detaches the aggregate OMVS.PRV.AGGR001.LDS0001.
```
zfsadm detach -aggregate omvs.prv.aggr001.lsd0001
```

IOEZ00122I Aggregate OMVS.PRV.AGGR001.LDS0001 detached successfully

**Related Information**
Commands:
- **zfsadm attach**

Chapter 12. zFS commands 141
zfsadm detach

Files:
  IOEFSPRM
zfsadm format

Purpose
Formats a VSAM Linear Data Set (VSAM LDS) as a zFS aggregate.

Format

Options
-aggregate name
Specifies the aggregate name of the aggregate to be formatted. This will be the name of the zFS aggregate that is formatted. The aggregate name is not case sensitive. It is translated to upper case.

-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.

Requirement: The default is not -compat. You should always specify -compat on the zfsadm format command.

-group {gid | name}
Specifies the group owner of the root directory of the file system. This is used with the -compat option, otherwise it is ignored. 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 8K 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 specified with this option are ignored.

-initialempty blocks
Specifies the number of 8K blocks that will be left empty at the beginning of the aggregate. The default is 1. If you specify 0, you will get 1 block. This option is not normally specified.

-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.

-logsize blocks
Specifies the number of 8K blocks reserved for the aggregate log. The default is 1% of the aggregate size or 128 megabytes, whichever is smaller. This 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.

-owner {uid | name}
Specifies the owner of the root directory of the file system. This is used with the -compat option, otherwise it is ignored. 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.
zfsadm format

-perms number
Specifies the permissions of the root directory of the file system. This is used with the 
-compat option, otherwise it is ignored. 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 will be lost. 
Use this option with caution. This option is not usually specified.

-size blocks
Specifies the number of 8K 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 LDS. 
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 LDS is attempted after 
the primary allocation is formatted unless the -grow option is specified. In that case, 
multiple extensions of the amount specified in the -grow option will be attempted until the 
-size is satisfied. Space must be available on the volume.

-system system name
Specifies the system that the format request will be sent to.

Usage
The zfsadm format command formats a VSAM LDS as a zFS aggregate. All zFS aggregates must be 
formatted before use (including HFS compatibility mode aggregates). The zfsadm format 
command requires the zFS PFS to be active on the system. The size of the aggregate is as many 8K blocks as fits 
in the primary allocation of the VSAM LDS 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 formatted 
and the size includes all of that space.

Privilege Required
The issuer of the zfsadm format command must have ALTER authority to the VSAM LDS and must be 
UID 0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the z/OS UNIXPRIV class.

Examples
The following command formats the VSAM LDS as a compatibility mode aggregate.
zfsadm format -aggregate omvs.prev.aggr001.lds0001 -compat -owner usera -group audit -perms o750

Related Information
Commands:
   zfsadm define

Files:
   IOEFSPRM
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 aggregate name of the aggregate to be grown. The aggregate name is not case sensitive. It is always translated to upper case.

-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.

-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)\(^4\) boundary. If zero is specified, the secondary allocation size will be used. The value specified cannot not 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 IDAMS LISTCAT ALL command. For an explanation of the rules that apply to extending a VSAM LDS, see z/OS DFSMS Using Data Sets, SC26-7410.

For a compatibility mode aggregate, the size of the file system quota will be increased by the amount of additional space available. For a multi-file system aggregate, the size of the file system quotas is not changed.

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 profile 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 profile in the z/OS UNIXPRIV class.

---

4. A Control Area is normally a cylinder or less and is based on the primary and secondary allocation units. See z/OS DFSMS Using Data Sets, SC26-7410 for more information about allocation size.
**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

Purpose
Shows syntax of specified zfsadm commands or lists functional descriptions of all zfsadm commands.

Format
zfsadm help [-topic command...] [-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 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.

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 IOΕFSPRM file in your zFS PROC, the issuer must have READ authority to the data set that contains the IOΕFSPRM file. If you are using parmlib (IOΕPRMxx), 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
**zfsadm lsaggr**

### Purpose
Lists all currently attached aggregates for zFS. The owning system is displayed in a shared file system (sysplex) environment.

### Format
```
zfsadm lsaggr [-system system name] [-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.

- **-system system name**
  Specifies the name of the system that owns the attached aggregates to be displayed.

### Usage
The **zfsadm lsaggr** command displays information about all attached aggregates.

This command 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 entire sysplex when all systems are running z/OS V1R7 and above):
```
zfsadm lsaggr
OMVS.PRV.AGGR004.LDS0004 JS000END R/W
OMVS.PRV.AGGR003.LDS0002 JS000END R/O
OMVS.PRV.AGGR003.LDS0001 JS000END R/W
OMVS.PRV.AGGR002.LDS0002 JS000END R/W
OMVS.PRV.AGGR001.LDS0001 JS000END R/W
```

### Related Information
**Command:**
- **zfsadm aggrinfo**

**File:**
- **IOEFSPRM**
**zfsadm lsfs**

**Purpose**
Lists all the file systems on a given aggregate or all attached aggregates.

**Format**

```
zfsadm lsfs [-aggregate name]  [-system system name] [-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 upper case. 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 if it contains one or more file systems or a message indicating that there are no file systems contained in the aggregate.

- **-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.

- **-long**
  Causes the output of the command to be extended to display the following additional information about space usage in an file system: the allocation limit, the quota 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 system name**
  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 an aggregate. The file systems do not need to be mounted to use this command.

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 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).
- If it is mounted or not.
- The allocation usage and the quota usage, in kilobytes.
- If the file system is on-line or not.
- If the file system is being cloned or if the backup is being deleted.
- The total number of file systems on-line, off-line, 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 is displayed:

- The total number of file systems contained in the aggregate.
The file system's name.
- The file system's ID.
- The type (RW for read-write, or BK for backup).
- If it is mounted or not.
- The state vector of the file system.
- If the file system is on-line or not.
- If the file system is being cloned or if the backup is being deleted.
- The allocation limit and allocation usage.
- The quota limit and quota usage.
- The size of the Filesystem Inode Table and the number of file requests.
- The version of the file system
- The day, date, and time when the file system was created (backed up for a backup file system).
- The day, date, and time when the contents of the file system were last updated (same as the creation time for a backup file system).
- The total number of file systems on-line, off-line, 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 2 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

OMVS.PRV.FS2 100000,,6 RW (Not Mounted) states 0x10010005 On-line
4294967232 K alloc limit; 9 K alloc usage
45000 K quota limit; 9 K quota usage
8 K Filesystem Inode Table 0 file requests
version 1.4
Creation Thu Aug 9 17:26:54 2001
Last Update Thu Aug 9 17:26:54 2001

Total file systems on-line 2; total off-line 0; total busy 0; total mounted 0
```

Related Information
Commands:
- zfsadm create
- zfsadm clone
zfsadm lsquota

Purpose
Shows quota information about file systems and aggregates.

Format
zfsadm lsquota {-filesystem name | -mfilesystem mount_name} [-aggregate name] [-level] [-help]

Options
-aggregate name
Specifies the name of the aggregate where the zFS file system name 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 upper case.

filesystem name
Specifies the name of the zFS file system about which quota and usage information is to be displayed. The file system name is case sensitive. If it was specified in upper case when the file system was created, it must be specified in upper case here.

-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.

-mfilesystem mount_name
Specifies the z/OS UNIX file system name of the file system about which quota and usage information is to be displayed. The file system name is case sensitive. If it was specified in upper case when the file system was mounted, it must be specified in upper case here.

Usage
The zfsadm lsquota command displays quota and usage information about a file system. The command also provides usage information about the aggregate in which the file system resides. The file system does not need to be mounted to use this command. The aggregate containing the file system must be attached.

The zfsadm lsquota command displays the name of the file system, the quota and the quota used (in kilobytes) of the file system, and the percentage of the quota in use. It also displays the information about the percentage of the aggregate in use, the number of kilobytes in use on the aggregate and the number of available kilobytes on the aggregate in which the file system resides. It also reports that the file system is zFS.

The size of a compatibility mode file system is equal to the size of the aggregate on which it resides. Therefore, the size and usage information displayed for the aggregate in the output of the zfsadm lsquota command equals the quota and quota usage information of the file system in the aggregate.

This command displays the following information about each specified file system:
- The name of the file system.
- The quota, in kilobytes, of the file system.
- The number of kilobytes of the quota currently in use on the file system.
- The percentage of the quota currently in use on the file system.
- The percentage of available disk space currently in use on the aggregate on which the file system resides.
The number of kilobytes of disk space in use on the aggregate and the total number of kilobytes on the aggregate on which the file system resides.

The file system type of the aggregate (zFS).

If the file system quota usage rises above 90% or the aggregate usage rises above 97%, the appropriate percentage is indicated with << and the message <<WARNING is displayed after the aggregate usage information at the end of the output line. (The 90% and the 97% are not related to the FSFULL and AGGRFULL options on MOUNT and in the IOEFSPRM file. Those are used to determine when to report to the operator.)

Note: Because each compatibility mode aggregate contains a single file system, the information displayed for a compatibility mode aggregate applies to the single file system it houses.

The `zfsadm aggrinfo` command can be used to display the total disk space on an aggregate and the amount currently available.

Every newly created zFS file system has a quota specification. The `zfsadm setquota` command can be used to increase or decrease the quota of a zFS file system. Because the quota of a zFS file system does not represent the amount of physical data space allocated to the file system, it can be larger than the size of the aggregate on which the file system resides. Similarly, the combined quotas of all file systems on an aggregate can be larger than the size of the aggregate. It cannot be changed to smaller than the usage of the file 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 command that follows lists quota and usage information for the file system `OMVS.PRV.FS1`. It also displays the size and usage information for the aggregate that contains this file system.

```
zfsadm lsq OMVS.PRV.FS1
```

```
Filesys Name Quota Used Percent Used Aggregate
OMVS.PRV.FS1 25000 9 0 1 = 1891/177992 (zFS)
```

The following command lists quota and usage information for the zFS file system named `OMVS.PRV.AGGR004.LDS0004`, and size and usage information for the aggregate on which the file system resides. The <<WARNING message directs the issuer's attention to the fact that the percentage of the quota in use on the indicated file system is above the warning level of 90% or the aggregate usage is above 97%.

```
zfsadm lsq -f OMVS.PRV.AGGR004.LDS0004
```

```
Filesys Name Quota Used Percent Used Aggregate
OMVS.PRV.AGGR004.LDS0004 1300 1266 97 <<100<< = 1412/1412 (zFS) <<WARNING
```

Related Information

Commands:
- `zfsadm aggrinfo`
- `zfsadm lsfs`
- `zfsadm setquota`
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 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

IOEZ0036I 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.
- `-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.
- `-iocounts` Specifies that the I/O count report should be displayed.
- `-knps` 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 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 -knps -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 system name` 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.
- `-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 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 is one of the queries that displays performance statistics.

```bash
zfsadm query -iobyaggr
```

<table>
<thead>
<tr>
<th>DASD</th>
<th>PAV</th>
<th>VOLSER</th>
<th>I/Os 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></td>
<td></td>
<td>CFC000</td>
<td>1 R/W</td>
<td>13</td>
<td>92</td>
<td>7641</td>
<td>30564</td>
<td>PLEX.JMS.AGGR001.LDS0001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CFC000</td>
<td>1 R/O</td>
<td>9</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>PLEX.JMS.AGGR002.LDS0002</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CFC000</td>
<td>1 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>
<td>--------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>48</td>
<td>340</td>
<td>12124</td>
<td>48516</td>
</tr>
</tbody>
</table>

Total number of waits for I/O: 52
Average I/O wait time: 3.886 (msecs)

Related Information
Commands:
```bash
zfsadm lsaggr
```
zfsadm quiesce

**Purpose**
Specifies that an aggregate and all the file systems 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 upper case. 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.

- **-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.

**Usage**
The **zfsadm quiesce** command is used to temporarily drain activity to the aggregate. During this time:

- No file systems in the aggregate can be created, deleted, renamed, or cloned.
- No quotas for file systems contained in the aggregate can be modified.
- 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, it 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 aggregate remains quiesced, message IOEZ00581E is displayed on the zFS owning system’s console.

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 profile 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 profile in the z/OS UNIXPRIV class.

**Examples**
The following command quiesces the aggregate OMVS.PRV.AGGR001.LDS0001.

```bash
zfsadm quiesce -aggregate omvs.prv.aggr001.lds0001
```

IOEZ00163I Aggregate OMVS.PRV.AGGR001.LDS0001 successfully quiesced
Related Information

Commands:
  zfsadm unquiesce
**Purpose**

Renames a file system. This is for multi-file system aggregates only. If you want to rename a compatibility mode aggregate see [Renaming or deleting a compatibility mode aggregate](#) on page 23.

**Format**

```
zfsadm rename -oldname oldname -newname newname [-aggregate name][-level] [-help]
```

**Options**

- **-aggregate name**
  Specifies the name of the aggregate where the zFS file system name resides. It is specified to qualify the zFS file system name (`-oldname`) 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 upper case.

- **-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.

- **-newname newname**
  Specifies the new zFS file system name for the read-write file system. The name must be unique within the sysplex (or system, if not in a sysplex), unique within the aggregate, and it should indicate of the file system's contents. The following characters can be included in the name of a file system:
  - 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).

  The name can be no longer than 44 characters. This length includes the `.bak` extension, which is added automatically when a read-only or backup version of the file system is created. If you intend to clone this file system, you might want to limit the file system name to 40 characters. Note that the `.bak` extensions are reserved for use with backup zFS file systems, so you cannot specify a file system name that ends with that extension.

  **Note:** The file system name is case sensitive. That is, if you specify the file system name in lower case as the `-newname` on the `zfsadm rename` command, you must specify the file system name in lower case when you mount it. The TSO/E MOUNT command translates the file system name to upper case even if it is within quotation marks. It is not translated to upper case if you specify the file system name on the TSO/E MOUNT command within triple quotation marks. For example, you can specify FILESYSTEM(""lower.case.example"") and the file system name is not translated to upper case. However, you might find it simpler to specify the file system name in upper case on the `zfsadm rename` command.

- **-oldname oldname**
  Specifies the current zFS file system name of the read-write file system. It is case sensitive.
Usage
The `zfsadm rename` command changes the name of the read-write file system specified with the `oldname` to the name specified with the `newname`. The name of the read-write file system's backup copy, if any, automatically changes to match. The aggregate that the file system is contained in must be attached. The file system cannot be mounted.

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 profile 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 profile in the z/OS UNIXPRIV class.

Examples
The following command changes the file system name `OMVS.PRV.FS2` to the file system name `OMVS.PRV.FS9`:
```
zfsadm rename -oldname OMVS.PRV.FS2 -newname OMVS.PRV.FS9
```

IOEZ00108I File system OMVS.PRV.FS2 renamed to OMVS.PRV.FS9
IOEZ00108I File system OMVS.PRV.FS2.bak renamed to OMVS.PRV.FS9.bak

Related Information
Commands:
- `zfsadm create`
- `zfsadm clone`
zfsadm setauditfid

Purpose
Sets (or resets) the zFS auditfid in the mounted aggregate.

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).

-**-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 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.

-**-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 will be set to binary zeros. The aggregate must be owned on an z/OS V1R9 or later system. Auditids for files and directories in file systems contained in multi-file system aggregates are not unique. Do not begin to use the unique auditid capability until all sysplex members are at z/OS V1R9.

Privilege Required
The user must be UID 0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the z/OS UNIXPRIV class.

Examples
```
zfsadm setauditfid -aggregate OMVS.PRV.AGGR001.LDS0001 -force
```

Related Information
Commands:
- `zfsadm aggrinfo`

File:
- IOEFSPRM
**zfsadm setquota**

**Purpose**
Sets the quota for a file system. This is for multi-file system aggregates only.

**Format**

```
zfsadm setquota {-filesystem name | -mfilesystem mount_name} -size kbytes [ -aggregate name] [-level] [-help]
```

**Options**

- `-aggregate name` Specifies the name of the aggregate where the zFS file system name 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 upper case.

- `-filesystem name` Specifies the file system name of the read-write file system whose quota is to be set. The file system name is case sensitive.

- `-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.

- `-mfilesystem mount_name` Specifies the z/OS UNIX file system name of the file system which the quota is to be set. The file system name is case sensitive. If it was specified in upper case when the file system was created, it must be specified in upper case here.

- `-size kbytes` Specifies the maximum amount of disk space that all of the files and directories in the read-write file system can occupy. This includes files and directories in the read-write version of the file system that are actually pointers to disk blocks in the backup version of the file system. Specify the value in 1-kilobyte blocks. (A value of 1024 kilobytes is 1 megabyte.) The minimum specification is 128 (that is, 128K bytes).

**Usage**

The `zfsadm setquota` command sets the quota limit for a read-write zFS file system. (It cannot be used to set the quota for a non-zFS file system or for a backup zFS file system.) The file system whose quota is to be set is indicated by specifying the file system name with the `-filesystem` option.

Quota refers to the amount of disk space occupied by all of the files and directories in the read-write version of the file system. This includes files and directories in the read-write version of the file system that are actually pointers to disk blocks in the backup version of the file system. Do not confuse quota with allocation; the latter identifies the amount of disk space occupied by the data that a file system actually houses; excluding those files and directories that are pointers to disk blocks in the backup version of the file system.

This command increases or decreases a file system’s quota to be the number of kilobytes specified with the `-size` option. Because it does not represent the amount of physical data the file system contains, a file system’s quota can be larger than the size of the aggregate on which it resides. Similarly, the sum of the quotas of all file systems on an aggregate can exceed the size of the aggregate.

The `zfsadm lsfs` and `zfsadm lsquota` commands display, among other things, the current quota for a file system.
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 profile 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 profile in the z/OS UNIXPRIV class.

Examples
The following command sets the quota for the file system named OMVS.PR.V.FC1 to be 15,000 kilobytes:

```
zfsadm setquota -filesystem OMVS.PR.V.FC1 -size 15000
```

```
zfsadm lsquota OMVS.PR.V.FC1
```

<table>
<thead>
<tr>
<th>Filesys Name</th>
<th>Quota</th>
<th>Used</th>
<th>Percent</th>
<th>Aggregate</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMVS.PR.V.FC1</td>
<td>15000</td>
<td>9</td>
<td>0</td>
<td>1 = 1907/177992 (zFS)</td>
</tr>
</tbody>
</table>

Related Information
Commands:
  zfsadm lsfs
  zfsadm lsquota
zfsadm unquiesce

Purpose
Makes an aggregate (and all the file systems contained in the aggregate) available to be accessed.

Format
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 upper case. 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 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.

Usage
The zfsadm unquiesce command allows activity that has been 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.

Privilege Required
The issuer must have READ authority to the data set that contains the IOEFSRPM file and must be logged in as root or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the z/OS UNIXPRIV class. If you are not using IOEFSRPM 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 profile in the z/OS UNIXPRIV class.

Examples
The following command unquiesces the aggregate OMVS.PRV.AGGR001.LDS0001.
zfsadm unquiesce -aggregate omvs.prv.aggr001.lds0001

IOE00166I Aggregate OMVS.PRV.AGGR001.LDS0001 successfully unquiesced

Related Information
Command:
zfsadm quiesce
zfsadm unquiesce
Chapter 13. zFS data sets

The following data sets are used during zFS processing.
**IOEFSPRM**

**Purpose**

This file lists the processing options for the zFS PFS and the definitions of the multi-file system aggregates. There is no mandatory information in this file, therefore it is not required. The options all have defaults. Aggregates can all be compatibility mode aggregates (which do not need definitions). Multi-file system aggregates can be attached by using the `zfsadm attach` command. They do not need definitions in IOEFSPRM to be attached using `zfsadm attach`. However, if you need to specify any options (for tuning purposes, for example) or if you want to have any multi-file system aggregates automatically attached when zFS is started, you need to have an IOEFSPRM file.

The location of the IOEFSPRM file is specified by the IOEZPRM DD statement in the zFS PROC. The IOEFSPRM file is typically a PDS member, so the IOEZPRM DD might look like the following:

```
//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.MSGOUT` results in `USERA.SY1.ZFS.MSGOUT` on system SY1 and `define_aggr cluster(USERA.&SYSNAME..AGGR001)` results in `define_aggr cluster(USERA.SY1.AGGR001)` 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 upper or lower case. Blank lines are allowed. You should not have any sequence numbers in the IOEFSPRM file. If you specify an invalid text value, the default value will be assigned. If you specify an invalid numeric value, and it is smaller than the minimum allowed value, the minimum value will be assigned. If you specify an invalid numeric value, and it is larger than the maximum allowed value, the maximum value will be assigned.

**Using PARMLIB (IOEPRMxx)**

The preferred alternative to the IOEZPRM DDNAME specification is specifying the IOEFSPRM member as a true PARMLIB member. In this case, the member has the name IOEPRMxx, where xx is specified in the parmlib member list.

When the IOEFSPRM is specified in the IOEZPRM DD statement of the zFS PROC, there can only be one IOEFSPRM file for each member of a sysplex. Using PARMLIB, zFS configuration options can be specified in a list of configuration parm 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, define_aggr) 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](https://www.ibm.com/support/knowledgecenter/SSLTBK_2.2.7/SIGcmp02f/rbparmlib_rref_oem.htm).

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 IOEPRMMyy members which contain zFS configuration statements. Columns 72-80 are ignored in the IOEPRMMyy member. The yy's are specified in the PARM option of the FILESYSTYPE statement for zFS (in the BPXPRMxx). The
only valid value that can be specified on the PARM option for zFS is the parmlib search parameter PRM=. The PARM string is case sensitive. You must enter the string in upper case. For example,

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

Up to 32 member suffixes can be specified. You can also use any system symbol that resolves to two characters. For example,

FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSCM)
ASNAME(ZFS,'SUB=MSTR')
PARM('PRM=(01,&SYSCLONE.)')

If &SYSCLONE.=AB, this specifies that parmlib member IOEPRMAB should be searched after parmlib member IOEPRM01. 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.

If no PRM suffix list is specified (and no IOEZPRM DD is specified in the zFS PROC), then member IOEPRM00 is read. PARMLIB support is only used when the IOEZPRM DD statement is not specified in the zFS PROC. When a IOEZPRM DD is specified in the zFS PROC, the single IOEFSPRM file specified in the DD is used, as previously.

To specify 32 members, type member suffixes up to column 71 and then continue them in column 1 on the next line. For example,

```
FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSCM) ASNAME(ZFS,'SUB=MSTR')
PARM('PRM=(00,01,02,03,04,05,06,07,08,09,10,11,12,13,14,
15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31)')
```

Coexistence between IOEFSPRM and IOEPRMxx

The FILESYSTYPE PARM PRM specification is ignored in releases prior to z/OS V1R6. Therefore, if you specify it in a BPXPRMxx member that is shared between z/OS V1R6 and previous releases (and no IOEZPRM DD is specified in the zFS PROC), the PRM specification is honored in z/OS V1R6 but is ignored in previous releases. This means that the IOEPRMxx members are searched for zFS configuration parameters in z/OS V1R6, but defaults are taken in previous releases. Also, if the ZFS FILESYSTYPE PARM has no PRM specification (and no IOEZPRM DD is specified in the zFS PROC), then zFS attempts to use the IOEPRM00 member in z/OS V1R6, but takes defaults in the previous releases. If IOEPRM00 is not found, then defaults are used.

Usage

The following options are used as processing options for the zFS PFS:

adm_threads
  Specifies the number of threads defined to handle pfsctl or mount requests.
  Default Value  10
  Expected Value  A number in the range of 1 - 256.
  Example  adm_threads=5

aggrfull
  Specifies the threshold and increment for reporting aggregate full error messages to the operator.
  The aggrfull parameter is independent of fsfull. However, aggrfull reports based on free 8 K blocks whereas, fsfull reports based on free 1 K blocks. aggrfull tends to give a more accurate view of free space and is the recommended choice.
aggrfull

Specifies whether aggregates can be dynamically extended when they become full. The aggregate (that is, the VSAM Linear Data Set) must have a secondary allocation specified to be dynamically extended and there must be space on the volume(s). This global value can be overridden in the define_aggr option or the zfsadm attach command for multi-file system aggregates and 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.

<table>
<thead>
<tr>
<th>Default Value</th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Value</td>
<td>Two numbers in the range of 1 - 99 within parentheses separated by a comma.</td>
</tr>
<tr>
<td>Example</td>
<td>aggrfull(90,5)</td>
</tr>
</tbody>
</table>

auto_attach

Controls whether aggregates defined and listed in the IOEFSRPM file are attached by default when zFS is started (or restarted). When the value is on, you can add new multi-file system aggregates (with the define_aggr option) to the IOEFSRPM file and they are attached automatically the next time zFS is started.

<table>
<thead>
<tr>
<th>Default Value</th>
<th>On</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Value</td>
<td>On or off</td>
</tr>
<tr>
<td>Example</td>
<td>auto_attach=on</td>
</tr>
</tbody>
</table>

client_cache_size

Specifies the amount of storage 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 [“user_cache_size” on page 171].

<table>
<thead>
<tr>
<th>Default Value</th>
<th>128M (only meaningful when zFS is running sysplex-aware)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Value</td>
<td>A number in the range of 10M - 65536M. K or M can qualify the number.</td>
</tr>
<tr>
<td>Example</td>
<td>client_cache_size=256M</td>
</tr>
</tbody>
</table>

client_reply_storage

Specifies the amount of storage used to handle sysplex server replies.

<table>
<thead>
<tr>
<th>Default Value</th>
<th>40M (only meaningful when zFS is running sysplex-aware)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Value</td>
<td>A number in the range of 2M - 128M. K or M can qualify the number.</td>
</tr>
<tr>
<td>Example</td>
<td>client_reply_storage=8M</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Default Value</th>
<th>Off</th>
</tr>
</thead>
</table>
Expected Value
On or off

Example
count_auditfid=on

dir_cache_size
Specifies the size of the directory buffer cache.
Default Value
32M
Expected Value
A number in the range of 2M - 512M
Example
dir_cache_size=32M

file_threads
Specifies the number of threads that handle sysplex server requests.
Default Value
40 (only meaningful when zFS is running sysplex-aware)
Expected Value
A number in the range of 1 - 256.
Example
file_threads=50

fsfull
Specifies the threshold and increment for reporting file system quota full error messages to the operator. The fsfull parameter is independent of aggrfull. Whereas aggrfull reports based on free 8K blocks whereas, fsfull reports based on free 1K 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 of 1 - 99 within parentheses separated by a comma.
Example
fsfull(85,5)

fsgrow
Specifies whether file systems in multi-file system aggregates can have their quota be dynamically extended when they reach their quota limit. The first number specifies the number of k-bytes that the file system quota should be increased. The second number specifies the number of times the quota should be extended. This global value can be overridden on the MOUNT command for multi-file system aggregates.
Default Value
Off
Expected Value
Two numbers in the range of 0 - 2147483648, within parentheses and separated by a comma.
Example
fsgrow(100,16)

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 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 detail, see the GRPNAME parameter of the IXCJOIN macro in 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 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. Note, the fixed option reserves real storage for usage by zFS only.
Default Value
16M
Expected Value
A number in the range of 2M - 1024M. A 'K' or 'M' can be appended to the value to mean kilobytes or megabytes, respectively.
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. Note, the fixed option reserves real storage for usage by zFS only.

Default Value: 32M
Expected Value: A number in the range of 1M - 1024M. 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 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.

Default Value: None
Expected Value: A number in the range of 1M - 2048M. A 'K' or 'M' can be appended to the value to mean kilobytes or megabytes, respectively.

Example: metaback_cache_size=64M,fixed

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, you should not specify this option.) It is read when zFS 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

nbs
Controls whether new block security is globally on by default or off by default for any aggregate. New block security refers to the guarantee made when a system fails. See "zfsadm attach" on page 121 for an explanation of the nbs option.

Default Value: On
Expected Value: On or off

Example: nbs=on

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 needs to 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: 256M
Expected Value: A number in the range of 128M - 512M.

Example: recovery_max_storage=128M

sync_interval
Specifies the number of seconds between syncs.
Default Value: 30
Expected Value
A number in the range of 11 - 21474836.

Example
sync_interval=45

sysplex
Specifies whether zFS should run in sysplex-aware mode. When sysplex=off, zFS will not automatically move the aggregate.

Default Value
OFF

Expected Value
On or off if BPXPRMxx specifies SYSPLEX(YES). Off if BPXPRMxx does not specify SYSPLEX(YES).

Example
sysplex=on

sysplex_admin_level
Specifies the zFS XCF communication interface level that zFS is running with in sysplex-aware mode. This is only valid for systems running zFS V1R9 or V1R10 with the proper APARs applied.

See [Chapter 2, “zFS post installation processing,” on page 11](#). The sysplex_admin_level option is ignored in z/OS V1R11 because zFS runs at sysplex_admin_level 3 on z/OS V1R11.

One (1) indicates that zFS running on z/OS V1R9 or V1R10 is in preconditioning mode and is using interface level 1. zFS uses the existing XCF protocol but also supports enough of the new XCF protocol for zFS interface level 2 to run properly. Two (2) indicates that zFS running on z/OS V1R9 or V1R10 is in toleration mode and is using interface level 2. zFS uses the new XCF protocol and is able to communicate with other zFS instances running at interface level 1 to display aggregate information for aggregates owned by zFS interface level 1 systems. At level 2, zFS is able to communicate with other zFS instances running at z/OS V1R11 (level 3). The value must be 2 on all members of the sysplex in order to bring z/OS V1R11 zFS into the shared file system environment. See [Chapter 2, “zFS post installation processing,” on page 11](#) for information on migrating to z/OS V1R11 zFS.

Default Value
3 in z/OS V1R11. 1 in z/OS V1R9 and V1R10

Expected Value
In z/OS V1R11, this option is ignored. In z/OS V1R9 and V1R10, 1 or 2.

Example
sysplex_admin_level=2

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" on page 172) when running in a shared file system environment and sysplex-aware, 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 of 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 of 200 - 10000000.

Example tran_cache_size=4000

user_cache_readahead
Specifies whether zFS does read ahead for sequential access. This is typically left on.

Default Value
On

Expected Value
On or off

Example user_cache_readahead=off

user_cache_size
Specifies the size, in bytes, of the cache used to contain file data. 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.

Default Value: 256 MB
Expected Value: A number in the range of 10 MB - 65536 MB (64G). A '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.

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 option is used to define multi-file system aggregates so that they are attached at zFS start (or restart):

define_aggr
Attaches a compatibility mode aggregate, specifying its corresponding data set name (which is the same as the aggregate name), and any processing suboptions for that aggregate. The define_aggr option can be contained on multiple lines and is complete when the next option is encountered or the end of the file is reached.

In z/OS V1R11 and above, you cannot attach a zFS multi-file system aggregate in a sysplex shared file system environment. You can attach a zFS compatibility mode aggregate (a zFS aggregate that contains one read-write file system and possibly a backup (.bak) file system) in any environment. Suboptions for define_aggr include:

aggrfull
Specifies the threshold and increment for reporting aggregate full messages for this aggregate to the operator. The default is the global aggrfull option (see 167).

aggrgrow or noaggrgrow
Indicates whether the multi-file system aggregate should dynamically grow when it runs out of physical space. The aggregate (that is, the VSAM Linear Data Set) must have secondary allocation specified and there must be space on the volume(s). The default is the global aggrgrow option (see 168).

attach or noattach
Indicates whether this aggregate is attached automatically when zFS is started (or restarted). Aggregates that are not automatically attached must be attached with the zfsadm attach command. The default is the global auto_attach option (see 168).

cluster
Specifies the VSAM Linear Data Set Cluster name. This is also the aggregate name. This option is required.

nbs or nonbs
Indicates if new block security algorithms should be used for this aggregate. The default is the global nbs option (see 170).

R/O or R/W
R/O specifies that the aggregate should be opened in read-only mode. A R/O aggregate means that all file systems in the aggregate are read-only and can only be mounted read-only. This allows sharing of an aggregate among multiple systems. R/W specifies that the aggregate should be opened in read-write mode. R/W is the default.

Example: define_aggr R/W attach nonbs aggrfull(85,5) aggrgrow cluster(OMVS.PRIV.AGGR0001.LDS0001)

The following options are used during debugging of the zFS PFS:
debug_settings_dsn
Specifies the name of a data set containing debug classes to enable when zFS starts up. It is read
when zFS is started (or restarted).
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)

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 9, “Performance and debugging,” on page 57. 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
Contains the output of any operator MODIFY ZFS,TRACE,PRINT commands or the trace output if
the zFS PFS abends. Each trace output creates a member in the PDSE. Traces that come from
the zFS PFS kernel have member names of ZFSKNTnn. nn starts with 01 and increments for
each trace output. nn is reset to 01 when zFS is started (or restarted). See Chapter 9,
“Performance and debugging,” on page 57. This is not a required parameter.
Default Value None
Expected Value The name of a PDSE data set.
Example trace_dsn=usera.zfs.trace.out

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 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.
Note: You can set the trace_table_size up to 2048M, but to print the trace to a PDSE you must
limit its size to 750M.
Default Value 16M
Expected Value A number in the range of 1M - 2048M.
Example trace_table_size=1M

xcf_trace_table_size
Specifies the size of the XCF trace table.
Default Value 4M
Expected Value A number in the range of 1M - 2048M.
Example xcf_trace_table_size=8M

The next two options are obsolete in z/OS Version 1 Release 3 and later and are ignored.
storage_details is always on and output from the MODIFY ZFS,QUERY, STORAGE,DETAILS goes into
the system log. storage_details_dsn is not used.

storage_details
Indicates whether the zFS internal storage tracking mechanisms are active. The results can be
sent to the storage_details_dsn with the operator MODIFY ZFS,QUERY,STORAGE,DETAILS
command.
Default Value Off
Expected Value On or off.
Example storage_details=on

storage_details_dsn
Indicates where the storage map is written if storage_details is on and the operator MODIFY
ZFS,QUERY,STORAGE,DETAILS command is run.
Default Value None
Expected Value The name of a data set.
IOEFSPRM

Example

storage_details_dsn=usera.zfs.storage.output(file1)

Examples

The following IOEFSPRM sample file lists every program option.

+ + + + Beginning of sample file + + + +

************************************************************************
* 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 (e.g. J0H232J) 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
*auto_attach=on
*client_cache_size=128M
*client_reply_storage=40M
*convert_auditfid=off
*dir_cache_size=32M
*file_threads=40
*fsfull(85,5)
*fgrow(100,16)
*group=IOEZFS1
*log_cache_size=32M
*meta_cache_size=64M
*metaback_cache_size=64M
*nbs=on
*romount_recovery=off
*recovery_max_storage=128M
*sync_interval=45
*sysplex=off
*token_cache_size=65536
*tran_cache_size=4000
*user_cache_readahead=off
*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 14. zFS application programming interfaces

This section contains programming interface information.

This chapter describes the zFS Application Programming Interface (API), pfsctl (BPX1PCT). It describes the ZFS commands: ZFSCALL_AGGR (0x40000005), ZFSCALL_FILESYS (0x40000004), ZFSCALL_CONFIG (0x40000006) and ZFSCALL_STATS (0x40000007) and their subcommands. These APIs are used to manage zFS aggregates and file systems and to query and set configuration options.

In z/OS V1R8 and above, the z/OS UNIX pfsctl (command X'C000000B') can also retrieve zFS reason code text. For additional 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.
pfsctl (BPX1PCT)

Purpose
The pfsctl (BPX1PCT) application programming interface is used to send physical file system specific requests to a physical file system. It is documented in a general manner in the z/OS UNIX System Services Programming: Assembler Callable Services Reference. ZFS is a physical file system and supports several (ZFS specific) pfsctl functions. These are documented in this section.

Format
BPX1PCT (File_system_type, Command, Argument_Length, Argument, Return_value, Return_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_AGGR (0x40000005)
- ZFSCALLFILESYS (0x40000004)
- ZFSCALL_CONFIG (0x40000006)
- ZFSCALL_STATS (0x40000007)

Each of these commands has a set of subcommands. The general format of the Argument for all subcommands is:

Subcommand operation code int
Parameter0 int
Parameter1 int
Parameter2 int
Parameter3 int
Parameter4 int
Parameter5 int
Parameter6 int
Buffer[n] char[n]

where n depends on the particular subcommand.

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.

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 document for these codes.
Reason Code
An integer in which the reason code is stored. If this code is of the form 0xEFnnxxxx, see z/OS
Distributed File Service Messages and Codes. Otherwise, see z/OS UNIX System Services
Messages and Codes.

Usage
There are four major commands: ZFSCALL_AGGR (0x40000005) and its subcommands,
ZFSCALL_FILESYS (0x40000004) and its subcommands, ZFSCALL_CONFIG (0x40000006) and
ZFSCALL_STATS (0x40000007) and its subcommands. zFS pfsctl APIs do not work across sysplex
members. zFS pfsctl APIs can query and set information on zFS aggregates owned by the current system
only. File system information from other systems will not show up. However, if all systems are running
z/OS Version 1 Release 7 and above, zFS pfsctl APIs will work across sysplex members.

Note: In z/OS V1R7 and above, the z/OS UNIX pfsctl (command X'C000000B') can also retrieve zFS
reason code text. For additional 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.

Aggregate commands
The Aggregate command code is ZFSCALL_AGGR (0x40000005). The following subcommands and their
subcommand opcodes are supported:
- Attach Aggregate (105)
- Create File System (131)
- Define Aggregate (139)
- Delete File System (136)
- Detach Aggregate (104)
- Format Aggregate (134)
- Grow Aggregate (129)
- List Aggregate Status (137)
- List Aggregate Status (Version 2) (146)
- List Attached Aggregate Names (135)
- List Attached Aggregate Names (Version 2) (140)
- List File System Names (138)
- List File System Names (Version 2) (144)
- Quiesce Aggregate (132)
- Set Auditorid (149)
- Unquiesce Aggregate (133).

File System commands
The File System command code is ZFSCALL_FILESYS (0x40000004). The following subcommands and
their subcommand opcodes are supported:
- Clone File System (143)
- List File System Status (142)
- Rename File System (140)
- Set File System Quota (141).

Configuration commands
The Configuration command code is ZFSCALL_CONFIG (0x40000006). The following subcommands and
their subcommand opcodes are supported:
- List Systems (174)
- Query adm_threads setting (180)
- Query aggrfull setting (181)
- Query aggrgrow setting (182)
- Query auto_attach setting (183)
- Query client_cache_size (231)
- Query client_reply_storage (223)
- Query cmd_trace (184)
- Query convert_auditfid (237)
- Query debug_settings_dsn setting (186)
- Query file_threads (217)
- Query fsfull setting (187)
- Query fsgrow setting (188)
- Query group setting (214)
- Query log_cache_size setting (193)
- Query meta_cache_size setting (198)
- Query metaback_cache_size setting (199)
- Query msg_input_dsn setting (200)
- Query msg_output_dsn setting (201)
- Query nbs setting (202)
- Query sync_interval setting (205)
- Query sysplex_state (215)
- Query syslevel (238)
- Query token_cache_size (216)
- Query trace_dsn setting (206)
- Query trace_table_size setting (207)
- Query tran_cache_size setting (208)
- Query user_cache_readahead setting (209)
- Query user_cache_size setting (210)
- Query vnode_cache_size setting (212)
- Set adm_threads (150)
- Set aggrfull (158)
- Set aggrgrow (171)
- Set client_cache_size (230)
- Set client_reply_storage (222)
- Set convert_auditfid (236)
- Set file_threads (176)
- Set fsfull (157)
- Set fsgrow (172)
- Set log_cache_size (153)
- Set meta_cache_size (152)
- Set metaback_cache_size (163)
- Set msg_output_dsn (161)
- Set nbs (156)
- Set sync_interval (154)
- Set token_cache_size (177)
- Set trace_dsn (159)
- Set tran_cache_size (160)
- Set user_cache_readahead (162)
- Set user_cache_size (151)
- Set vnode_cache_size (155)

### Statistics commands

The statistics command code is ZFSCALL_STATS (0x40000007). The following subcommands and their subcommand codes are supported:

- Statistics directory cache information (249)
- Statistics iobyaggr information (244)
- Statistics iobydasd information (245)
- Statistics iocounts information (243)
- Statistics kernel information (246)
- Statistics locking information (240)
- Statistics log cache information (247)
- Statistics metadata cache information (248)
• Statistics storage information (241)
• Statistics transaction cache information (250)
• Statistics user data cache information (242)
• Statistics vnode cache information (251)
Attach Aggregate

Purpose
The Attach Aggregate subcommand call is an aggregate operation that attaches a multi-file system aggregate to a system. This makes the aggregate and all its file systems known to the ZFS Physical File System running on that system.

Format
syscall_parmlist

opcode 105 AGOP_ATTACH_PARMDATA
parms[0] offset to AGGR_ID
parms[1] offset to AGGR_ATTACH
parms[2] offset to system name (optional)
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_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_RO 0x40 Attach aggregate as read-only
ATT_NBS 0x20 Use New Block Security
ATT_NONBS 0x10 Do not use new block security
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
EMVSERR Internal error using an osi service
EPERM Permission denied to perform request

Reason_code
0xEFnnxxxx See z/OS Distributed File Service Messages and Codes

Usage
This function is used to attach multi-file system aggregates. Compatibility mode aggregates are attached during mount so that a separate attach is not necessary.

ATT_NBS and ATT_NONBS are mutually exclusive. If neither is specified, the default is the nbs setting in the IOEFSRPM file. See "zfsadm attach" on page 121 for a description of the nbs parameter.
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 21 and “Dynamically growing a multi-file system aggregate” on page 53 for a description of dynamic grow.

at_threshold and at_increment are ignored unless ATT_MONITOR is set.

Reserved fields and undefined flags must be set to binary zeros.

Offset to systemname in parms[2] can be specified in z/OS V1R7 and above. The systemname can only refer to a system running z/OS V1R7 and above.

**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**

Delete 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 */
#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 struct aggr_attach_t {
    char at_eye[4]; /* Eye catcher */
#define AT_EYE "AGAT"
    short at_len; /* Length of structure */
    char at_ver; /* Structure version */
#define AT_VER_INITIAL 1 /* Version 1 */
    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_RO 0x40 /* aggr should be attached ro */
#define ATT_NBS 0x20 /* aggr should be attached with full NBS */
#define ATT_NONBS 0x10 /* aggr should be attached with no NBS */
#define ATT_GROW 0x04 /* allow dynamic grow */
#define ATT_NOGROW 0x02 /* disallow dynamic grow */
char at_res2; /* Reserved for future use */
int at_reserved[64]; /* Reserved for future use */
char systemname[9];

struct parmstruct
{
syscall_parmlist myparms;
AGGR_ID aggr_id;
AGGR_ATTACH myaggr;
char systemname[9]; /* System to attach on */
};

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;

AGGR_ID *idp = &(myparmstruct.aggr_id);
AGGR_ATTACH *atp = &(myparmstruct.myaggr);
char *asp = myparmstruct.systemname;

myparmstruct.myparms.opcode = AGOP_ATTACH_PARMDATA;
myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
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 the owner of the aggregate to be a different system than this one */
/* myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(AGGR_ID) + sizeof(AGGR_ATTACH); */
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 */
mmemset(atp,0,sizeof(AGGR_ATTACH)); /* Ensure reserved fields are 0 */
mmemset(asp,0,sizeof(myparmstruct.systemname)); /* 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;
strncpy(myparmstruct.aggr_id.aid_name,aggrname);
memcpy(&myparmstruct.myaggr.at_eye[0], AT_EYE, 4);
myparmstruct.myaggr.at_len = sizeof(AGGR_ATTACH);
myparmstruct.myaggr.at_ver = AT_VER_INITIAL;
myparmstruct.myaggr.at_threshold = 90; /* 90 percent threshold */
myparmstruct.myaggr.at_increment = 5; /* 5 percent increment */
myparmstruct.myaggr.at_flags = 0;
myparmstruct.myaggr.at_flags |= ATT_MONITOR; /* Use threshold and */
/* increment */
myparmstruct.myaggr.at_flags |= ATT_GROW; /* allow dynamic growing */
/* This next field should only be set if parms[2] is non-zero */
/* strcpy(myparmstruct.systemname,"DCEIMGVQ"); */

BPXIPCT("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 attaching aggregate %s on system %s\n", aggrname,myparmstruct.systemname);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else /* Return from attach was successful */
{
    printf("Aggregate %s attached successfully on system %s\n",aggrname,myparmstruct.systemname);
}
return 0;
}
Clone File System

Purpose
The Clone File System subcommand call is a file system operation that creates (or replaces) a backup file system from the specified read-write file system. This is referred to as cloning a file system. The backup file system is stored in the same aggregate as the read-write file system.

You can use an FS_ID or an FS_ID2 as input.

Format
syscall_parmlist

opcode 143 FSOP_CLONE_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
  high unsigned long 0
  low unsigned 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_mtnname char[45] 0
fsid_reserved char[49] 0

Return_value 0 if request is successful, -1 if it is not successful

Return_code

EBUSY Aggregate containing file system is quiesced
EINVAL Invalid parameter list
EMVSERR Internal error using an osi service
ENOENT Aggregate is not attached
EPERM Permission denied to perform request
EROFS Aggregate is attached as read only

Reason_code

0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
Usage
The aggregate containing the read-write file system to be cloned must be attached. The backup file system name is the same as the read-write file system's name with .bak appended. After the clone operation, the backup file system can be mounted read-only.

After the backup file system is mounted read-only, users can access this point-in-time copy of the data until the backup file system is deleted or the read-write file system is recloned.

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
Delete File System

Restrictions
The aggregate cannot be attached as read-only. The file system name of the read-write file system to be cloned must be less than or equal to 40 characters. If the backup file system already exists, it cannot be mounted. The aggregate containing the read-write file system cannot be quiesced. You cannot specify both a file system name and a mount file system name.

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
Example 1 - Using FS_ID

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_FILESYS 0x40000004
#define FSOP_CLONE_PARMDATA 143

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;
    #define FSID_VER INITIAL 1  /* Length of this structure */
    char fsid_ver;
    /* Version */
    #define FSID_RES1 1  /* Initial version */
    char fsid_res1;
    /* Reserved. */
} fs_id_t;
```
Clone File System

```c
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 = FSOP_CLONE_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_FILESYS, /* 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 cloning file system \%s\n", filesystemname);
        printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
        return bpxrc;
    }
    else /* Return from clone file system was successful */
    {
        printf("File system \%s cloned successfully\n", filesystemname);
    }
    return 0;
}

Example 2 - Using FS_ID2

#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
```
```c
#include <stdio.h>

#define ZFSCALL_FILESYS 0x40000004
#define FSOP_CLONE_PARMDATA 143

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_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;
    FS_ID2 fsid;
} ;

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char filesystemname[45] = "OMVS.PRV.FS3";

    struct parmstruct myparmstruct;

    FS_ID2 *idp = &(myparmstruct.fsid);
```
Clone File System

myparmstruct.myparms.opcode = FSOP_CLONE_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",
    ZFS Call_FILESYS, /* 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 cloning file system %s\n", filesystemname);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
    return bpxrc;
}
else /* Return from clone file system was successful */
{
    printf("File system %s cloned successfully\n", filesystemname);
}
return 0;
}
Create File System

Purpose

The Create File System subcommand call is an aggregate operation that creates a new read-write file system in a multi-file system aggregate on a system.

You can use an FS_ID or an FS_ID2 as input.

Format

syscall_parmlist

opcode 131  AGOP_CREATEFILESYS_PARMDATA
params[0]  offset to FS_ID or FS_ID2
params[1]  offset to FILESYS_DATA
params[2]  0
params[3]  0
params[4]  0
params[5]  0
params[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]  "OMVS.PRV.AGGR001.LDS0001"
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

FILESYS_DATA

fd_eye  char[4]  "FSDT"
fd_len  short  sizeof(FILESYS_DATA)
fd_ver  char  1
fd_flags  char
  FD_OWNER_SPECIFIED  0x80
  FD_PERMS_SPECIFIED  0x40
fd_owner  int  612
fd_group  int  10
fd_perms  int  0755
fd_quotah  short  0
fd_reserved1  char[2]  0
fd_quota  unsigned long  5000
fdReserved  char[64]  0

Return_value  0 if request is successful, -1 if it is not successful

Return_code
Create File System

EBUSY  Aggregate containing file system is quiesced
EXIST  File system already exists
EINTR  ZFS is shutting down
EMVSERR Internal error using an osi service
EPERM  Permission denied to perform request
EROFS  Aggregate is attached as read only

Reason_code
0xEFnnxxxx  See z/OS Distributed File Service Messages and Codes

Usage
The aggregate that is to contain the new read-write file system must be attached. The file system name can be no longer than 44 characters. If this file system is to be cloned, a file system name extension of .bak will be added to the end of the read-write file system name to create the backup file system name. If you intend to clone this read-write file system, you might want to limit the read-write file system name to 40 characters.

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
Clone File System
Delete File System

Restrictions
The aggregate cannot be quiesced or attached as read-only. You cannot create a file system that already exists. You cannot create a file system that ends with .bak. The fd_quota must be at least 128 (for 128 K bytes).

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).

Examples
Example 1 - Using FS_ID

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_CREATEFILESYS_PARMDATA 131

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 */
  char fsid_len;       /* Length of this structure */
  char fsid_ver;       /* 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 fsidReserved2[2];                  /* Reserved for the future */
} FS_ID;

typedef struct filesys_t {
  char fd_eye[4];    /* eye catcher */
  short fd_len;      /* Length */
  char fd_ver;       /* */
  char fd_flags;     /* Flag bits */
  char fd_owner;     /* Owner id for root */
  char fd_group;     /* Group id for root */
  char fd_perms;     /* Permissions for root */
  short fd_quotah;   /* High portion of quota, in K bytes */
  unsigned long fd_quota; /* Low portion of quota in K bytes */
  char fd_reserved[64]; /* More reserved bytes */
} FILESYS_DATA;

struct parmstruct {
  syscall_parmlist myparms;
  FS_ID fsid;
  FILESYS_DATA myfilesystem;
};

int main(int argc, char **argv) {
  int bpxrv;
  int bpxrc;
  int bpxrs;
  char filesystemname[45] = "OMVS.PRV.FS3";
  char aggrname[45] = "OMVS.PRV.AGGR001.LDS0001";

  struct parmstruct myparmstruct;

  FS_ID *idp = &(myparmstruct.fsid);
  FILESYS_DATA *fdp = &(myparmstruct.myfilesystem);

  myparmstruct.myparms.opcode = AGOP_CREATEFILESYS_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;

  "Create File System"

Chapter 14. zFS application programming interfaces 193
Create File System

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(fdp,0,sizeof(FILESYS_DATA)); /* 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_aggrname,aggrname);
strcpy(myparmstruct.fsid.fsid_name,filesystemname);

memcpy(&myparmstruct.myfilesystem.fd_eye[0], FD_EYE, 4);
myparmstruct.myfilesystem.fd_len = sizeof(FILESYS_DATA);
myparmstruct.myfilesystem.fd_ver = FD_VER_INITIAL;
myparmstruct.myfilesystem.fd_flags = FD_OWNER_SPECIFIED | FD_PERMS_SPECIFIED;
myparmstruct.myfilesystem.fd_owner = 612;
myparmstruct.myfilesystem.fd_group = 10;
myparmstruct.myfilesystem.fd_perms = 0755; /* permissions (in octal) */
myparmstruct.myfilesystem.fd_quotah = 0;
myparmstruct.myfilesystem.fd_quota = 5000; /* Size of file system in K-bytes */

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 creating file system %s in aggregate %s\n",filesystemname,aggrname);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else /* Return from create file system was successful */
{
    printf("File system %s in Aggregate %s created successfully\n",filesystemname,aggrname);
}
return 0;

Example 2 - Using FS_ID2

Example 2 - Using FS_ID2

#include <stdio.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_CREATEFILESYS_PARMDATA 131

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

194  z/OS V1R11.0 Distributed File Service zFS Administration
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_mtnname[ZFS_MAX_FSYSNAME+1]; /* Mount name, null terminated */
    char fsid_reserved[49]; /* Reserved for the future */
} FS_ID2;

typedef struct filesys_t {
    char fd_eye[4]; /* eye catcher */
#define FD_EYE "FSDT"
    short fd_len; /* Length */
    char fd_ver; /* */
#define FD_VER_INITIAL 1 /* Initial version */
    char fd_flags; /* Flag bits */
#define FD_OWNER_SPECIFIED 0x80 /* Owner & group specified */
#define FD_PERMS_SPECIFIED 0x40 /* Permissions specified */
    int fd_owner; /* Owner id for root */
    /* filesystem */
    int fd_group; /* Group id for root */
    /* filesystem */
    int fd_perms; /* Permissions for root */
    /* filesystem */
#define FD_DEFAULT_PERMS 0755 /* Default permissions if not specified */
    short fd_quotah; /* High portion of quota, in */
    /* K bytes */
    char fd_reserved1[2]; /* Reserved bytes */
    unsigned long fd_quota; /* Low portion of quota in */
    /* K bytes */
    char fd_reserved[64]; /* More reserved bytes */
} FILESYS_DATA;

struct parmstruct
{
    syscall_parmlist myparms;
    FS_ID2 fsid;
    FILESYS_DATA myfilesystem;
};

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
Create File System

```c
int bpxrv;
char filesystemname[45] = "OMVS.PRVS.FS3";
char aggrname[45] = "OMVS.PRVS.AGGR001.LDS0001";

struct parmstruct myparmstruct;

FS_ID2 *idp = &(myparmstruct.fsid);
FILESYS_DATA *fdp = &(myparmstruct.myfilesystem);

myparmstruct.myparms.opcode = AGOP_CREATEFILESYS_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 */
memez(fd0,0,sizeof(FILESYS_DATA)); /* 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_aggrname,aggrname);
strcpy(myparmstruct.fsid.fsid_name,filesystemname);

memcpy(&myparmstruct.myfilesystem.fd_eye[0], FD_EYE, 4);
myparmstruct.myfilesystem.fd_len = sizeof(FILESYS_DATA);
myparmstruct.myfilesystem.fd_ver = FD_VER_INITIAL;
myparmstruct.myfilesystem.fd_flags = FD_OWNER_SPECIFIED | FD_PERMS_SPECIFIED;
myparmstruct.myfilesystem.fd_owner = 612;
myparmstruct.myfilesystem.fd_group = 10;
myparmstruct.myfilesystem.fd_perms = 0755; /* permissions (in octal) */
myparmstruct.myfilesystem.fd_quotah = 0;
myparmstruct.myfilesystem.fd_quota = 5000; /* Size of file system in K-bytes */

BPX1PCT("ZFS ",
ZFCALL_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 creating file system %s in aggregate %s\n",filesystemname,aggrname);
    printf("BPRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else /* Return from create file system was successful */
{
    printf("File system %s in Aggregate %s created successfully\n",filesystemname,aggrname);
}

return 0;
```

}}

196  z/OS V1R11.0 Distributed File Service zFS Administration
Define Aggregate

Purpose
The Define Aggregate subcommand call is an aggregate operation that defines (creates) a VSAM Linear
Data Set (VSAM LDS). This VSAM LDS can then be formatted as a zFS aggregate.

Format
syscall_parmlist
    opcode 139 AGOP_DEFINE_PARMDATA
    parms[0] offset to AGGR_DEFINE
    parms[1] size of Buffer
    parms[2] offset to Buffer
    parms[3] offset to system name (optional)
    parms[4] 0
    parms[5] 0
    parms[6] 0

    AGGR_DEFINE
    an_eye char[4] "AGDF"
    an_len char sizeof(AGGR_DEFINE)
    an_ver char 1
    an_aggrName char[45] "OMVS.PRV.AGGR001.LDS0001"
    an_dataClass char[9] 0
    an_managementClass char[9] 0
    an_storageClass char[9] 0
    an_model char[45] 0
    an_modelCatalog char[45] 0
    an_volumes[59] char[7] "PRV000"
    an_reservedChars1 char 0
    an_numVolumes int 1
    an_spaceUnit int 1 /* 1 = cylinders */
    an_spacePrimary int 10 /* 10 cylinders */
    an_spaceSecondary int 1 /* 1 cylinder */
    an_reservedInts1 char[32] 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 parameters
    EMVSERR Internal error using an osi service
    ENOENT Aggregate is not attached
    EPERM Permission denied to perform request

Reason_code
    OxEFnnxxxx See z/OS Distributed File Service Messages and Codes

Usage
Reserved fields and undefined flags must be set to binary zeros.

Privilege Required
The issuer must have sufficient authority to create the VSAM LDS.

Related Services
    Format Aggregate
Restrictions
The VSAM LDS to be defined cannot already exist

Examples

```c
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, char *, int *, int *, int *, int *);

#include <stdio.h>
#define ZFS_CALL_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;

#define ZFS_MAX_AGGR_NAME 44
#define ZFS_MAX_SMS_ID 8
#define ZFS_MAX_VOL_ID 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 ver; /* Version */
    char aggrName[ZFS_MAX_AGGR_NAME+1];
    char dataClass[ZFS_MAX_SMS_ID+1];
    char managementClass[ZFS_MAX_SMS_ID+1];
    char storageClass[ZFS_MAX_SMS_ID+1];
    char model[ZFS_MAX_AGGR_NAME+1];
    char modelCatalog[ZFS_MAX_AGGR_NAME+1];
    char catalog[ZFS_MAX_AGGR_NAME+1];
    char volumes[59][ZFS_MAX_VOL_ID+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 bpivr;
    int bpxrc;
    int bpxrs;
    char aggrname[45] = "PLEX.JMS.AGGR007.LDS0007"; /* aggregate name to define */
    char dataclass[9] = "";
```
Define Aggregate

```c
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_PARMDATA;
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 */

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 the define to run on a different system than this one */

/* myparmstruct.myparms.parms[3] = */
/* myparmstruct.myparms.parms[0]+sizeof(AGGR_DEFINE)+sizeof(myparmstruct.Buffer); */

myparmstruct.myparms.parms[4] = 0;
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(agp,0,sizeof(*agp));
strcpy(agp->eye,ADEF_EYE);
agp->ver=ADEF_VER_INITIAL;
agp->len=sizeof(AGGR_DEFINE);
memset(bufp,0,sizeof(myparmstruct.Buffer));

strcpy(agp->aggrName,aggrname);
strcpy(agp->model,model); /* If included next 4 can be null */
strcpy(agp->dataClass,modelcatalog);
strcpy(agp->managementClass,managementclass);
strcpy(agp->storageClass,storageclass);
strcpy(agp->modelCatalog,modelcatalog);
strcpy(agp->volumes[0],(char *)volumes);
agp->numVolumes=1;
agp->spaceUnit=ZFS_SPACE_CYLS;
agp->spacePrimary=10;
agp->spaceSecondary=1;

BPX1PCT("ZFS",
    ZFSCALL_AGGR,
    sizeof(myparmstruct),
    (char *)&myparmstruct,
    &bpxr, &bpxr, &bpxr);

if (bpxr < 0) {
    printf("define: Error defining LDS %s\n",aggrname);
    printf("define: BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxr,bpxr,bpxrs);
    printf("define: job output:\n\n\n",myparmstruct.Buffer);
    return bpxrc;
}
```
Define Aggregate

else{
    printf("define: LDS %s defined successfully\n",aggrname);
}
return 0;
Delete File System

Purpose
The Delete File System subcommand call is an aggregate operation that deletes an existing read-write file system from a multi-file system aggregate on a system. It can also be used to delete an existing backup file system.

You can use an FS_ID or an FS_ID2 as input.

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 hyper
    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

Return_code
  EBUSY Aggregate containing file system is quiesced
  EXIST File system does not exist
  EINTR ZFS is shutting down
  EMWSERR Internal error using an osi service
  EPERM Permission denied to perform request
  EROFS Aggregate is attached as read only

Reason_code
  0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
Delete File System

Usage
The aggregate that contains the file system to be deleted must be attached. Read-write file systems and backup file systems are related during removal as follows:

- Removing a read-write file system automatically removes its associated backup version (if the backup version exists).
- Removing a backup file system does not remove the 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
- Clone File System
- Create File System

Restrictions
The aggregate cannot be quiesced or attached as read-only. You cannot delete a file system that is mounted. If you are removing a read-write file system and it has a backup file system, neither the read-write nor the backup file systems can be mounted.

- When using an FS_ID2 as input, you cannot specify the file system with the z/OS UNIX file system name (fsid_mtnname) since the file system cannot be mounted. You must use the zFS file system name (fsid_name).
- 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
Example 1 - Using FS_ID

```c
#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"
} fs_id_t;
```
# Delete File System

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 */
        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;
}

Example 2 - Using FS_ID2
Delete File System

#include <stdio.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_DELETEFILESYS_PARM_DATA 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[49]; /* 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.PRIV.FS3";

    struct parmstruct myparmstruct;
}

204 z/OS V1R11.0 Distributed File Service zFS Administration
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
The Detach Aggregate subcommand call is an aggregate operation that detaches a multi-file system aggregate from a system. This makes the aggregate and all its file systems unavailable to the ZFS Physical File System running on that system.

Format
syscall_parmlist

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(opcode)</td>
<td>104 AGOP_DETACH_PARMDATA</td>
</tr>
<tr>
<td>parms[0]</td>
<td>offset to AGGR_ID</td>
</tr>
<tr>
<td>parms[1]</td>
<td>0</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>
<tr>
<td>AGGR_ID</td>
<td>aid_eye char[4] &quot;AGID&quot;</td>
</tr>
<tr>
<td></td>
<td>aid_len char sizeof(AGGR_ID)</td>
</tr>
<tr>
<td></td>
<td>aid_ver char 1</td>
</tr>
<tr>
<td></td>
<td>aid_name char[45] &quot;OMVS.PRV.AGGR001.LDS0001&quot;</td>
</tr>
<tr>
<td></td>
<td>aid_reserved char[33] 0</td>
</tr>
</tbody>
</table>

Return_value 0 if request is successful, -1 if it is not successful

Return_code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBUSY</td>
<td>Aggregate could not be detached due to mounted file system</td>
</tr>
<tr>
<td>EINTR</td>
<td>ZFS is shutting down</td>
</tr>
<tr>
<td>EMVSERR</td>
<td>Internal error using an osi service</td>
</tr>
<tr>
<td>ENOENT</td>
<td>Aggregate is not attached</td>
</tr>
<tr>
<td>EPERM</td>
<td>Permission denied to perform request</td>
</tr>
</tbody>
</table>

Reason_code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xEFnnxxx</td>
<td>See z/OS Distributed File Service Messages and Codes</td>
</tr>
</tbody>
</table>

Usage
This function is used to detach multi-file system aggregates. Compatibility mode aggregates are detached during unmount.

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 */
    #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;
    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);
    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 */
```

Chapter 14. zFS application programming interfaces  207
Detach Aggregate

bspxrs);         /* Pointer to Reason_code */

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
The Format Aggregate subcommand call is an aggregate operation that formats a VSAM Linear Data Set (VSAM LDS) as a zFS aggregate.

Format
syscall_parmlist

opcode 134 AGOP_FORMAT_PARMDATA
parms[0] offset to AGGR_ID
parms[1] offset to AGGR_FORMAT
parms[2] offset to system name (optional)
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_FORMAT
af_eye char[4] "AGFM"
af_len short sizeof(AGGR_FORMAT)
af_ver char 1
af_res1 char 0
af_size long 0
af_logsize long 0
af_initialempty long 0 /* 0 gives 1 block */
af_overwrite int 0 /* Use caution if you specify 1 */
af_compat int 1 /* compat aggr desired */
af_owner int 0 /* no uid specified */
af_ownerSpecified int 0 /* use uid of issuer */
af_group int 0 /* no guid specified */
af_groupSpecified int 0 /* gid set to issuer default group */
af_perms int 0 /* no perms specified */
af_grow int 0 /* grow amount, 0 means grow not specified */
af_newauditfid int 0 /* old auditfid; l=newauditfid */
af_reserved char[56]
systemname char[9]

Return_value 0 if request is successful, -1 if it is not successful

Return_code
EBUSY Aggregate is busy or otherwise unavailable
EINTR ZFS is shutting down
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
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 profile in the z/OS UNIXPRIV class.

Related Services

Define Aggregate

Restrictions

The VSAM LDS 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 */
#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;

typedef struct aggr_format_t {
    char af_eye[4]; /* Eye catcher */
#define AF_EYE "AGFM"
    short af_len; /* Length of structure */
    char af_ver; /* Version of cb */
#define AF_VER_INITIAL 1 /* For future use */
    char af_res1; /* Amount to format of aggr */
#define AF_DEFAULT_SIZE 0 /* If set, we use default of entire primary partition of LDS */
    long af_size; /* Size of logfile in aggr */
#define AF_DEFAULT_LOGSIZE 0 /* If set, we use default of 1% of aggr size */
    long af_logsize; /* Initial empty blocks */
#define AF_DEFAULT_INITIALEMPTY 1 /* This is the default & minumum too */
    int af_initialempty;
#define AF_OVERWRITE_OFF 0 /* Overwrite aggr if its not empty */
    int af_overwrite; /* Overwrite off, that means if aggr not empty it will */
    /* NOT be formatted, th default */
#define AF_OVERWRITE_ON 1 /* Overwrite in effect */
    int af_compress; /* HFS-compag gr desired */
#define AF_MULT 0 /* Multi-file sys aggr desired */
#define AF_HFSCOMP 1 /* HFS-compag gr desired */
    int af_owner; /* Owner for HFS-compag */
    int af_ownerSpecified; /* Indicates an owner was provided */
```
```c
#define AF_OWNER_USECALLER 0  /* Owner gets set to pfsctl issuer uid */
#define AF_OWNER_SPECIFIED 1  /* Use owner uid set in af_owner */
int af_group; /* Group for HFS-compat */
int af_groupSpecified; /* Indicates if group specified */
#define AF_GROUP_USECALLER 0  /* Group gets set to pfsctl issuer default group */
#define AF_GROUP_SPECIFIED 1  /* Use group gid set in af_group */
#define AF_DEFAULT_PERMS 0755 /* The default perms to use */
#define AF_PERMS_DEFAULT 0 /* Perms not specified, use default */
#define AF_PERMS_SPECIFIED 1 /* Use perms set in af_perms */
int af_perms; /* Perms for HFS-compat */
int af_permsSpecified; /* Indicates if perms provided */
#define AF_DEFAULT_SIZE 1 /* I want an HFS compatibility mode aggregate */
int af_grow; /* Amount to extend each time until we reach desired size */
int af_newauditfid; /* 0 = old format auditfid, 1 = new format auditfid */
char af_reserved[56]; /* For future use */
} AGGR_FORMAT;

struct parmstruct {
    syscall_parmlist myparms;
    AGGR_ID aid;
    AGGR_FORMAT aggformat;
    char systemname[9];
} myparmstruct;

main(int argc, char **argv)
{
    int 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 */
    myparmstruct.myparms.opcode = AGOP_FORMAT_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist)+sizeof(AGGR_ID);
    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 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);

    memset(idp,0,sizeof(AGGR_ID));
    memset(aggptr,0,sizeof(myparmstruct.aggformat));
    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->af_eye,AF_EYE,4);
    aggptr->af_len = sizeof(myparmstruct.aggformat);
    aggptr->af_ver = AF_VER_INITIAL;
    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 */

    bpxrv = bpxrc = bpxrs = 0;
    exit(0);
}
```

Chapter 14. zFS application programming interfaces 211
aggptr->af_permsSpecified=AF_PERMS_DEFAULT; /* aggptr->af_perms = perms; */
aggptr->af_grow = 0; /* no grow size */
aggptr->af_newauditfid = 1; /* generate a new auditfid */  
ZFSAGGR, /* 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 BPRC = %d BPXS = %0x\n",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else {
    printf("Formatted aggregate %s\n",aggrname);
}
return 0;
Grow Aggregate

Purpose
The Grow Aggregate subcommand call is an aggregate operation that extends the physical size of an aggregate. It can also be used to extend compatibility mode aggregates and multi-file system aggregates.

Format
syscall_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
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
8 DFSMS did not extend the aggregate
EBUSY Aggregate containing file system is quiesced
EINTR ZFS is shutting down
EINVAL Invalid parameter list
EMVSERR Internal error using an osi service
ENOENT Aggregate is not attached
EPERM Permission denied to perform request
EROFS Aggregate is attached as read only

Reason_code
0xEFnnxxxx See z/OS Distributed File Service Messages and Codes

Usage
The aggregate to be grown must be attached. The size specified is the new total size (in 1 K-byte blocks) 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. The determination of whether to extend to another volume is made by DFSMS. 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.

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
List Aggregate Status
**Grow Aggregate**

**Restrictions**
The aggregate to be grown cannot already be quiesced and cannot be attached as read-only. An aggregate cannot be made smaller.

**Examples**
```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 */
    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.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 */
    &bpxrv, /* Pointer to Return_value */
    &bpxrc, /* Pointer to Return_code */
    &bpxrs); /* Pointer to Reason_code */

if (bpxrv < 0)
{
    printf("Error growing aggregate %s\n", aggrname);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
    return bpxrc;
}
else /* Return from grow was successful */
{
    printf("Aggregate %s grown successfully\n", aggrname);
}
return 0;
}
List Aggregate Status

Purpose
The List Aggregate Status subcommand call is an aggregate operation that returns information about a specified attached aggregate on this system.

Format
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.PR.V.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_res1  char  0
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
This call returns information about a specified aggregate. The aggregate must be attached.
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_PARMDATA 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_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]; /* aggr name, null terminated */
    char aid_reserved[33]; /* Reserved for the future */
} AGGR_ID;

typedef unsigned long u_long;

#define AID_EYE "AGID"
define AID_VER_INITIAL 1

define AGST
    short as_len;       /* Length of structure */
    char as_ver;        /* Initial version */
    char as_res1;       /* Reserved. */
    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 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_res2;       /* Reserved */
```

Chapter 14. zFS application programming interfaces
List Aggregate Status

```c
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_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;
AGGR_ID *idp = &(myparmstruct.aggr_id);
AGGR_STATUS *asp = &(myparmstruct.aggr_status);

myparmstruct.myparms.opcode = AGOP_GETSTATUS_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_STATUS)); /* Ensure reserved fields are 0 */
memcpy(&myparmstruct.aggr_status.as_eye[0], AS_EYE, 4);
myparmstruct.aggr_status.as_len = sizeof(AGGR_STATUS);
myparmstruct.aggr_status.as_ver = AS_VER_INITIAL;
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);

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 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=%x\n",
```
myparmstruct.aggr_status.as_aggrId,
myparmstruct.aggr_status.as_nFileSystems,
    myparmstruct.aggr_status.as_flags);

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);

return 0;
}
}
List Aggregate Status (Version 2)

Purpose
The List Aggregate Status subcommand call is an aggregate operation that returns information about a specified attached aggregate on this system. Version 2 returns additional flags and fields.

Format

```c
syscall_parmlist
  opcode 146 AGOP_GETSTATUS2_PARMDATA
  parms[0] offset to AGGR_ID
  parms[1] offset to AGGR_STATUS2
  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_STATUS2)
  as_ver char 2
  as_res1 char 0
  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_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] /* Reserved for future */
```

Return_value 0 if request is successful, -1 if it is not successful

Return_code

220 z/OS V1R11.0 Distributed File Service zFS Administration
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
This call returns information about a specified aggregate. The aggregate must be attached.

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
#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;

#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"
} aggr_status_t;
List Aggregate Status (Version 2)

short as_len; /* Length of structure */
char as_ver;
#define AS_VER_2 2 /* version 2 */
char as_res1; /* Reserved. */
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 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 */
char as_flags2; /* Aggregate flags2 */
#define AS_DISABLED 0x80 /* 1 = Aggr is disabled */

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) */
long as_totalUsable; /* Total available blocks on aggregate (normally 8K) */
long as_realFree; /* Total kilobytes free */
long as_minFree; /* Minimum kilobytes free */
int as_reserved2[3];
int as_freeblocks; /*Number of k available in free 8k blocks*/
int as_free frags; /*Number of k available in free 1k fragments*/
int as_directLog; /*Number of k used on the log*/
int as indirectLog; /*Number of k used indirectly on the log*/
int as_fstbl; /*Number of k used for the filesystem table*/
int as_bitmap; /*Number of k used for the bitmap file*/
int as_diskFormatMajorVersion; /* disk format major version */
int as_diskFormatMinorVersion; /* disk format minor version */
char as_auditfid[10]; /* 6 byte volser followed by 4 byte CCHH */
char as_bytes_reserved[2]; /* reserved */
char as_reserved[72]; /* 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;
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 = AGOP_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_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);

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 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: version=%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");
}
return 0;

List Attached Aggregate Names

Purpose
The List Attached Aggregate Names subcommand call is an aggregate operation that returns a list of the names of all attached aggregates on a system.

Format
```
syscall_parmlist
  opcode 135 AGOP_LISTAGGRNAMES_PARMDATA
  parms[0] buffer length or 0
  parms[1] offset to AGGR_ID 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[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]
```

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
E2BIG List is too big for buffer supplied
```

Usage
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
#include <stdio.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_LISTAGGRNAMES_PARMDATA 135
#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_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]; /* aggr name, null terminated */
    char aid_reserved[33]; /* Reserved for the future */
} AGGR_ID;

typedef struct parmstruct {
    syscall_parmlist myparms;  /* Real malloc'd structure will have an array of AGGR_IDs here */
    long size;
    char systemname[9];
} parmstruct;

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 */
              &bpxrv,
```
List Attached Aggregate Names

```c
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 */
    /* systemp = (char *)myp + buflen + sizeof(syscall_parmlist) + sizeof(long); */
    /* strcpy(systemp,"DCEIMGVN"); */ /* set system to get lsaggr info from */
    myp->myparms.opcode = AGOP_LISTAGGRNAMES_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[3] = sizeof(syscall_parmlist) + buflen + sizeof(long);

    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);
    if (bpxrv == 0)
    {
      total_aggrs = buflen/aggSize;
      count_aggrs = 1;
      for(aggPtr = (AGGR_ID *) &(myp->size) ; count_aggrs <= total_aggrs ;
          aggPtr++, count_aggrs++)
      {
        if (strlen(aggPtr->aid_name) != 0)
          printf("%-64.64s\n",aggPtr->aid_name);
      }
      free(myp);
    } /* lsaggr names failed with large enough buffer */
    else /* error was not E2BIG */
    {
      printf("Error on ls aggr trying to get required size\n");
      printf("BPRXRV = %d BPXRC = %d BPXRS = %lx\n",bpxrv,bpxrc,bpxrs);
      free(myp);
      return bpxrc;
    }
  } /* error was not E2BIG */
else /* error was not E2BIG */
{
  printf("Error on ls aggr with large enough buffer\n");
  printf("BPRXRV = %d BPXRC = %d BPXRS = %lx\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");
  }
}
return 0;
}
List Attached Aggregate Names (Version 2)

**Purpose**
The List Attached Aggregate Names subcommand call is an aggregate operation that returns a list of the names of all attached aggregates on a system with the system name.

**Format**
system_call_parmlist

<table>
<thead>
<tr>
<th>opcode</th>
<th>AGOP_LISTAGGRNAMES2_PARMDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>parms[0]</td>
<td>buffer length or 0</td>
</tr>
<tr>
<td>parms[1]</td>
<td>offset to AGGR_ID2 or 0</td>
</tr>
<tr>
<td>parms[2]</td>
<td>offset to size</td>
</tr>
<tr>
<td>parms[3]</td>
<td>offset to system name (optional)</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[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
ENOENT  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**
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_LISTAGGRNAMES2_PARMDATA 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_AGGRNAME 44
#define SYS_MAX_NAMELEN 8 /* Max. z/OS system name length*/

typedef struct aggr_id2_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]; /* 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 myparamstruct;
    AGGR_ID2 *aggPtr;
    int aggSize = sizeof(AGGR_ID2);
    int buflen = sizeof(AGGR_ID2);
    struct parmstruct *myp = &myparamstruct;
    int myssize;
    char *systemp;
    int count_aggrs, total_aggrs;

    myparamstruct.myparms.opcode = AGOP_LISTAGGRNAMES2_PARMDATA;
    myparamstruct.myparms.parms[0] = 0;
    myparamstruct.myparms.parms[1] = 0;
    myparamstruct.myparms.parms[2] = sizeof(syscall_parmlist);
    myparamstruct.myparms.parms[3] = 0;
    myparamstruct.myparms.parms[4] = 0;
    myparamstruct.myparms.parms[5] = 0;
    myparamstruct.myparms.parms[6] = 0;

    BPX1PCT("ZFS ",
        ZFSCALL_AGGR, /* Aggregate operation */
        sizeof(myparmstruct), /* Length of Argument */
```
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 */
    /* 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",
             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 */
  }
  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");
  }
}
return 0;
}
List File System Names

Purpose
The List File System Names subcommand call is an aggregate operation that returns the names of the file systems contained in a specified aggregate on this system.

Format
```c
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]
    Array of FS_IDS (n can be zero)
    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
- EINTR ZFS is shutting down
- EINVAL Invalid parameter list
- EMVSERR Internal error using an osi service
- ENOENT 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
The aggregate specified must be attached.

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
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL 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 */
    /* parms[4]-parms[6] are currently unused*/
} 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 */
    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 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 */
    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_reserved2[2]; /* Reserved for the future */
} FS_ID;

struct parmstruct {
    syscall_parmlist myparms;
    AGGR_ID aggr_id;
    /* Real malloc'd structure will have an array of FS_IDS here */
    long size;
};
```
int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;

    struct parmstruct myparmstruct;
    AGGR_ID *aggPtr;
    FS_ID *fsPtr;
    int fsSize = sizeof(FS_ID);
    int buflen = sizeof(FS_ID);
    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_PARMDATA;
    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 */
             &bpxrv, /* Pointer to Return_value */
             &bpxrc, /* Pointer to Return_code */
             &bpxrs); /* Pointer to Reason_code */

    if (bpxrv < 0)
    {
        if (bpxrc == E2BIG)
        {
            buflen = my->size; /* Get buffer size needed */
            mypsize = buflen + sizeof(syscall_parmlist) + sizeof(AGGR_ID) + sizeof(long);
            my = (struct parmstruct *) malloc ((long) mypsize);
            memset(my, 0, mypsize);
            memcpy(my->aggr_id.aid_eye,AID_EYE,4);
            my->aggr_id.aid_len = sizeof(AGGR_ID);
            my->aggr_id.aid_ver = AID_VER_INITIAL;
            strcpy(my->aggr_id.aid_name,aggrname);
            my->myparms.opcode = AGOP_LISTFSNAMES_PARMDATA;
            my->myparms.parms[0] = sizeof(syscall_parmlist);
            my->myparms.parms[1] = buflen;
            my->myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
            my->myparms.parms[3] = sizeof(syscall_parmlist) + sizeof(AGGR_ID) + buflen;
            my->myparms.parms[4] = 0;
            my->myparms.parms[5] = 0;
            my->myparms.parms[6] = 0;
            BPX1PCT("ZFS ",
                     ZFSCALL_AGGR, /* Aggregate operation */
                     mypsize, /* Length of Argument */
                     my->aggr_id.aid_name,/* Pointer to Argument */
                     &bpxrv, /* Pointer to Return_value */
                     &bpxrc, /* Pointer to Return_code */
                     &bpxrs); /* Pointer to Reason_code */
        }
    }
}
(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\n",total_fs);
    count_fs = 1;
    for(fsPtr = (FS_ID *) &(myp->size); count_fs <= total_fs; fsPtr++, count_fs++)
    {
        printf("%-64.64s\n",fsPtr->fsid_name);
    }
    free(myp);
} else /* lsaggr names failed with large enough buffer */
{
    printf("Error on ls fs 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 fs 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 file systems */
{
    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 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).

Format
syscall_parmlist
   opcode 144 AGOP_LISTFSNAMES_PARMDATA2
   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_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
   EMVSERR Internal error using an osi service
   ENOENT  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
The version 2 List File System Names returns an array of FS_ID2s.

The aggregate specified must be attached.

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_mtname), you cannot specify
the zFS file system name (fsid_name) nor the aggregate name (fsid_aggrname).

Examples

```c
#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;

#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 */
    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 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_mtname[ZFS_MAX_FSYSNAME+1];    /* Mount name, null terminated */
    char fsid_reserved[49];    /* Reserved for the future */
} FS_ID2;

struct parmstruct
{
```
List File System Names (Version 2)

syscall_parmlist mypars;
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 bpxrv;
    int bpxrc;
    int bpxrs;

    struct parmstruct myparmstruct;
    AGGR_ID *aggPtr;
    FS_ID2 *fsPtr;
    int fsSize = 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";

    long *p;

    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 */
        &bpxrv, /* Pointer to Return_value */
        &bpxrc, /* Pointer to Return_code */
        &bpxrs); /* Pointer to Reason_code */

    if (bpxrv < 0)
    {
        if (bpxrv == 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;

BPXIPCT("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_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_mname);
    }
    free(myp);
    }
else /* lsaggr names failed with large enough buffer */
{
    printf("Error on ls fs 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 fs 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 file systems */
{
    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

The List File System Status subcommand call is a file system operation that lists the status information of a file system.

You can use an FS_ID as input or (if you want to specify the z/OS UNIX file system name (that is, the mount name)) you can use an FS_ID2 as input. Of course, if you use the z/OS UNIX file system name, the file system must be mounted using that file system name.

Format

syscall_parmlist

opcode 142  FSOP_GETSTAT_PARMDATA
parms[0] offset to FS_ID
parms[1] offset ro 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 short sizeof(FS_ID)
fsid_ver char 1
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.PRIV.FS3"
fsid_reserved char[32] 0
fsid_reserved2 char[2] 0

FS_ID2 or FS_ID

fsid_eye char[4] "FSID"
fsid_len short 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] 0
fsid_mtnname char[45] "OMVS.PRIV.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
List File System Status

fs_accStatus long 0
fs_states long 0
fs_nodeMax 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]
fs_InodeTbl unsigned long
fs_requests
    high unsigned long
    low unsigned long
fs_reserved3 unsigned long
fs_reserved4 unsigned long
fs_reserved5 unsigned long
fs_pad1 int
fs_diskFormatMajorVersion unsigned long
fs_diskFormatMinorVersion unsigned long
fs_reserved char[80]

Return_value 0 if request is successful, -1 if it is not successful

Return_code
EBUSY Aggregate containing file system is quiesced
EINVAL Invalid parameter list
EINTR ZFS is shutting down
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
The aggregate containing the file system to be listed must be attached.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required
None.

Related Services
List Attached Aggregate Names
List File System Aggregate Names

Restrictions
The aggregate containing the file system to be listed cannot be quiesced.

| 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).
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).

The following fields are internal use only and are not intended for application usage.
- fs_accError
- fs_accStatus
- fs_type

The following 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.
- fs_states

Examples

Example 1 - Using 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_id_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_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_reserved2[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 {
```

List File System Status

242 z/OS V1R11.0 Distributed File Service zFS Administration
List File System Status

Chapter 14. zFS application programming interfaces 243

```c
char fs_eye[4];  /* Eye catcher */
#define FS_EYE "FSST"
short fs_len;  /* Length of structure */
char fs_ver;
#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 */
ulong fs.allocLimit;  /* Allocation limit for filesystem in kilobytes */
ulong fs.allocUsage;  /* Amount of allocation used in kilobytes */
ulong fs.visQuotaLimit;  /* Visible filesystem quota in kilobytes */
ulong fs.visQuotaUsage;  /* How much quota is used in kilobytes */
ulong 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 inode number used */
long fs.minQuota;
long fs.type;
char fs.threshold;  /* Threshold for fsfull monitoring */
char fs.increment;  /* Increment for fsfull monitoring */
char fs.mountstate;  /* Aggregate flags */
#define FS_NOT_MOUNTED 0  /* Filesys not mounted */
#define FS_MOUNTED_RW 1  /* Filesys mounted RW */
#define FS_MOUNTED_RO 2  /* Filesys mounted RO */
char fs_msglen;  /* Length of status message */
char fs_msg[128];  /* Status message for filesystem */
char fs.aggrname[ZFS_MAX_AGGRNAME+1];  /* Name of aggregate I reside on */
char fs.reserved1[3];  /* Reserved for future use/alignment */
unsigned long fsreserved2[3];  /* Reserved */
ulong fs.InodeTbl;  /* Amount of k used for the Filesystem Inode table */
hyper fs_requests;  /* Number of filesystem requests by users/applications */
ulong fs_reserved3;
ulong fs_reserved4;
ulong fs_reserved5;
int fs_pad1;
ulong fs.diskFormatMajorVersion;  /* Disk format major version */
ulong fs.diskFormatMinorVersion;  /* Disk format minor version */
char fs_reserved[80];  /* Reserved for future use */

) FS_STATUS;

struct parmstruct
{
    syscall_parmlist myparms;
    FS_ID fs_id;
    FS_STATUS fs_status;
};

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 */
memequ(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 ",
 ZFSCALL_FILESYS,
 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=%d, 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_aggrname);
}

List File System Status
myparmstruct.fs_status.fs_InodeTbl,
  myparmstruct.fs_status.fs_requests.high,
  myparmstruct.fs_status.fs_requests.low);
printf("getstatus: version=%d.%d\n",
  myparmstruct.fs_status.fs_diskFormatMajorVersion,
  myparmstruct.fs_status.fs_diskFormatMinorVersion);
}
return 0;
}

Example 2 - Using FS_ID2

#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_aggrname[ZFS_MAX_AGGRNAME+1]; /* Aggregate name, can be NULL string */
  char fsid_name[ZFS_MAX_FSYSNAME+1]; /* Name, null terminated */
  char fsid_mtname[ZFS_MAX_FSYSNAME+1]; /* Mount 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;
  #define FS_VER_INITIAL 1 /* Initial version */
  char fs_flags;
} fs_status_t;
List File System Status

#define FS_PERINFO 0x80 /* Performance information in output status */
hyper fs_id; /* Internal identifier */
struct timeval fs_cloneTime; /* Time when this file sys made via clone or when last recloned */
struct timeval fs_createTime; /* Time when this file sys was created */
struct timeval fs_updateTime; /* Time when this file sys was last updated */
struct timeval fs_accessTime; /* Time when this file sys was last accessed */
u_long fs_allocLimit; /* Allocation limit for file sys in kilobytes*/
u_long fs_allocUsage; /* Amount of allocation used in kilobytes*/
u_long fs_visQuotaLimit; /* Visible filesystem quota in kilobytes*/
u_long fs_visQuotaUsage; /* How much quota is used in kilobytes*/
lon fs_accError; /* error to return for incompatible vnode ops */
lon fs_accStatus; /* Operations currently being performed on file system */
lon fs_states; /* State bits */
define FS_TYPE_RW 0x10000 /* read-write (ordinary) */
define FS_TYPE_BK 0x30000 /* `.backup' */
lon fs_nodeMax; /* Maximum inode number used */
lon fs_minQuota;
lon fs_type;
char fs_threshold; /* Threshold for fsfull monitoring */
char fs_increment; /* Increment for fsfull monitoring */
char fs_mountstate; /* Aggregate flags */
define FS_NOT_MOUNTED 0 /* File sys not mounted */
define FS_MOUNTED_RW 1 /* File sys mounted RW */
define FS_MOUNTED_RO 2 /* File sys mounted RO */
char fs_msglen; /* Length of status message */
char fs_msg[128]; /* Status message for file system */
char fs_aggrname[ZFS_MAX_AGGRNAME+1]; /* Name of aggregate I reside on */
char fs_reserved1[3]; /* Reserved for future use/alignment */
unsigned lon fs_reserved2[3]; /* reserved */
u_long fs_inodeTbl; /*Amount of k used for the Filesystem Inode table*/
/*fs_inodeTbl 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;
int fs_pad1;
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
{
    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);
myparmstruct.fs_id2.fsid_ver = FSID_VER_2;
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",
            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=%d, 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_inodeTbl,
            myparmstruct.fs_status.fs_requests.high,
            myparmstruct.fs_status.fs_requests.low);
    printf("getstatus: version=%d,,%d\n",
            myparmstruct.fs_status.fs_diskFormatMajorVersion,
            myparmstruct.fs_status.fs_diskFormatMinorVersion);
List File System Status

}
List Systems

Purpose
The List Systems subcommand call is used to retrieve the system names that are part of the zFS XCF group.

Format
syscall_parmlist
    opcode  174  CFGOP_LSSYS
    parms[0]  size of buffer
    parms[1]  offset to buffer
    parms[2]  offset to size
    parms[3]  0
    parms[4]  0
    parms[5]  0
    parms[6]  0
buffer  char[ ]
size   int

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
    0xEFnnxxx  See z/OS Distributed File Service Messages and Codes

Usage
List Systems is used to retrieve the system names that are part of the zFS XCF group.

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 *);

    #include <stdio.h>

    #define ZFCALL_CONFIG  0x40000006
    #define CFGOP_LSSYS    174  /* List names of systems in the sysplex */
    #define E2BIG           145  /* data to return is too big for buffer */
    #define ERANGE         2    /* there were no systems to return */
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;

    BPX1PCT("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(syscall_parmlist) + 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] = sizeof(syscall_parmlist); /* offset to buffer */
            myp->myparms.parms[2] = sizeof(syscall_parmlist) + buflen; /* offset to size */
            myp->myparms.parms[3] = 0;
            myp->myparms.parms[4] = 0;
            myp->myparms.parms[5] = 0;
            myp->myparms.parms[6] = 0;

            BPX1PCT("ZFS ",
                ZFSCALL_CONFIG, /* Config query operation */

List Systems

250  z/OS V1R11.0 Distributed File Service zFS Administration
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\n", 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);
    }
}
List Systems

return 0;
Query Config Option

Purpose
The Query Config Option is a set of subcommand calls (that are configuration operations) that retrieve the current value for a particular configuration setting. Each one returns the particular configuration setting as a character string.

The following Format and Example use the CFGOP_QUERY_ADM_THREADS subcommand. The other query subcommands (see "Configuration commands" on page 179) operate in a similar manner. That is, each of them return the configuration setting as a character string in the co_string field.

Format
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
Query Config Option subcommands are used to retrieve the current value of a particular configuration option. Each subcommand retrieves one configuration as a character string.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required
None.

Related Services
Set Config Option
Query Config Option

Restrictions
None.

Examples

```c
#include <stdio.h>
#define ZFS_CALL_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 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 */
   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;

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); /* Only specify a non-zero offset for the next field (parms[1]) if */
    myparmstruct.myparms.parms[1] = 0; /* you are running z/OS 1.7 and above, and */
    myparmstruct.myparms.parms[2] = 0; /* you want to configure to a different system */
    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);

BPX1PCT("ZFS CONFIG", sizeof(myparmstruct), (char *) &myparmstruct, &bpxrv, &bpxrc, &bpxrs);
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;

#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 */

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 "CFOP"
    short co_len; /* Length 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;

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);

BPX1PCT("ZFS ",
ZFSCALL_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\n",bpxrv,bpxrc,bpxrs);
    return bpxrc;
} else {
    /* Parse our configquery string */
    version = myparmstruct.co.co_string;
    service = strchr(version, '\n');
    *service = '\0';
    service++;
    created = strchr(service, '\n');
    *created = '\0';
    created++;
    sysplex = strchr(created, '\n');
    *sysplex = '\0';
    sysplex++;
    interface = strchr(sysplex, '\n');
    *interface = '\0';
    interface++;
    sysplex_level = atoi(sysplex);
    if (sysplex_level == NO_SYSPLEX_SUPPORT)
        printf("zFS kernel: z/OS zSeries File System\nVersion %s Service Level %s.\nCreated on %s.\n",version,service,created);
    else
        char buffer[80];
        if (sysplex_level == SYSPLEX_ADMIN_LEVEL)
            sprintf(buffer, "sysplex(admin-only) interface(\%s)\", interface);
else /* if sysplex_level is SYSPLEX_FILE_LEVEL */
    sprintf(buffer, "sysplex(file) interface(%s)", interface);
    printf("zFS kernel: z/OS zSeries File System\nVersion %s Service Level %s.\nCreated \n
    on %s.\n%s
",version,service,created,buffer);
}
return 0;
Quiesce Aggregate

Purpose
The Quiesce Aggregate subcommand call is an aggregate operation that quiesces a compatibility mode aggregate or a multi-file system aggregate. This quiesces activity on the aggregate and all its file systems.

Format
syscall_parmlist
   opcode 132 AGOP_QUIESCE_PARMDATA
   parms[0] offset to AGGR_ID
   parms[1] offset to handle returned by quiesce
   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
   quiesce_handle long

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
Quiesce Aggregate is used to suspend activity on an aggregate. All activity on file systems contained in the aggregate that are mounted is also suspended. This is typically used before backing up an aggregate. The aggregate must be attached in order to be quiesced. The quiesce operation returns a quiesce handle that must be supplied on the unquiesce call.

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.myparms.opcode = AGOP_QUIESCE_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(&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);

    BPX1PCT("ZFS ",
        ZFSCALL_AGGR, /* Aggregate operation */
        sizeof(myparmstruct), /* Length of Argument */
```

Chapter 14. zFS application programming interfaces  259
Quiesce Aggregate

```c
(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 quiescing aggregate %s\n", aggrname);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
    return bpxrc;
}
else /* Return from quiesce was successful */
{
    printf("Aggregate %s quiesced successfully, quiescehandle=%d\n", aggrname, myparmstruct.quiesce_handle);
    save_quiesce_handle = myparmstruct.quiesce_handle;
}
return 0;
```
Rename File System

Purpose

The Rename File System subcommand call is a file system operation that renames a file system.

You can use an FS_ID or an FS_ID2 as input for the old zFS file system name or the new zFS file system name (that is, the fsid_name).

Format

syscall_parmlist

opcode 140  FSOP_RENAME_PARMDATA

parms[0] offset to FS_ID (Old Name)
parms[1] offset to FS_ID (New Name)
parms[2] 0
parms[3] 0
parms[4] 0
parms[5] 0
parms[6] 0

FS_ID or FS_ID2

/* Old File System Name */

fsid_len short sizeof(FS_ID)
fsid_ver char 1
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_reserved char[32] 0

FS_ID2 or FS_ID

/* Old File System Name */

fsid_len short 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_reserved char[32] 0

FS_ID or FS_ID2

/* New File System Name */

fsid_len short sizeof(FS_ID)
fsid_ver char 1
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.FS4”
fsid_reserved char[32] 0

FS_ID2 or FS_ID

/* New File System Name */

fsid_len short sizeof(FS_ID2)
fsid_ver char 2
fsid_res1 char 0
fsid_res2 char 0
**Rename File System**

```c
fsid_id
    high unsigned long 0
    low unsigned long 0
fsid_aggrname char[45] 0
fsid_name char[45] 0
fsid_mtnname char[45] "OMVS.PRIV.FS4"
fsid_reserved char[49] 0
```

**Return_value**
0 if request is successful, -1 if it is not successful

**Return_code**
- `EBUSY` Aggregate containing file system is quiesced
- `EINVAL` Invalid parameter list
- `EINTR` ZFS is shutting down
- `EMVSERR` Internal error using an osi service
- `ENOENT` Aggregate is not attached
- `EPERM` Permission denied to perform request
- `EROFS` Aggregate is attached as read only

**Reason_code**
0xEFnnxxxx See z/OS Distributed File Service Messages and Codes

**Usage**
The aggregate that contains the file system to be renamed must be attached.

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**
Clone File System

**Restrictions**
A backup file system cannot be renamed by itself. You must rename the read-write file system (which renames both the read-write and the backup file systems). You cannot rename a read-write file system to an name that ends in `.bak`. This is reserved for backup file systems. If a backup file system exists, you cannot rename a read-write file system to a name that is longer than 40 characters. The file system to be renamed cannot be mounted. The aggregate containing the file system to be renamed cannot be quiesced or attached as read-only. If you specify an aggregate name in the old file system name FS_ID and in the new file system name FS_ID, they must be the same.

When using an FS_ID2 for the old file system name, you cannot specify the z/OS UNIX file system name (fsid_mtnname) because the file system to be renamed cannot be mounted. When using the FS_ID2 for the new file system name, you cannot specify the z/OS UNIX file system name (fsid_mtnname) because the API needs the new zFS file system name.

Examples
Example 1 - Using FS_ID
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_FILESYS 0x40000004
#define FSOP_RENAME_PARMDATA 140

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 */
    /* FSID */
    char fsid_len; /* Length of this structure */
    char fsid_ver; /* Version */
    /* FSID_VER_INITIAL 1 */
    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_old;
    FS_ID fsid_new;
};

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char old_filesystemname[45] = "OMVS.PRV.FS3";
    char new_filesystemname[45] = "OMVS.PRV.FS4";

    struct parmstruct myparmstruct;
    FS_ID *idop = &(myparmstruct.fsid_old);
    FS_ID *idnp = &(myparmstruct.fsid_new);

    myparmstruct.myparms.opcode = FSOP_RENAME_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(idop,0,sizeof(FS_ID)); /* Ensure reserved fields are 0 */
memset(idnp,0,sizeof(FS_ID)); /* Ensure reserved fields are 0 */

memcpy(&myparmstruct.fs_id.old.fsid_eye, FSID_EYE, 4);
myparmstruct.fs_id.old.fsid_len = sizeof(FS_ID);
myparmstruct.fs_id.old.fsid_ver = FSID_VER_INITIAL;
strcpy(myparmstruct.fs_id.old.fsid_name,old_filesystemname);

memcpy(&myparmstruct.fs_id.new.fsid_eye, FSID_EYE, 4);
myparmstruct.fs_id.new.fsid_len = sizeof(FS_ID);
myparmstruct.fs_id.new.fsid_ver = FSID_VER_INITIAL;
strcpy(myparmstruct.fs_id.new.fsid_name,new_filesystemname);

BPX1PCT("ZFS ", /* File system operation */
ZFSCALL_FILESYS, /* Length of Argument */
sizeof(myparmstruct), /* Pointer to Argument */
(char *) &myparmstruct, /* Pointer to Return_value */
&bpxrv, /* Pointer to Return_code */
&bpxrc, /* Pointer to Reason_code */
&bpxrs);

if (bpxrv < 0)
{
    printf("Error renaming file system %s to %s\n",old_filesystemname, new_filesystemname);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else /* Return from rename file system was successful */
{
    printf("File system %s was renamed to %s successfully\n",old_filesystemname, new_filesystemname);
}

return 0;

Example 2 - Using FS_ID2

#include <stdio.h>

typedef int 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 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 ZFSCALL_FILESYS 0x40000004
#define FSOP_RENAME_PARM_DATA 140

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 */
    char fsid_len; /* Length of this structure */
    char fsid_ver; /* Initial version */
    char fsid_res1; /* Reserved. */
    char fsid_res2; /* Reserved. */
} fs_id_t;
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 R14 */
    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[ZFS_MAX_FSYSNAME+1]; /* Mount name, null terminated */
    char fsid_reserved[2]; /* Reserved for the future */
} FS_ID2;

struct parmstruct
{
    syscall_parmlist myparms;
    FS_ID2 fsid_old;
    FS_ID2 fsid_new;
};

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    char old_filesystemname[45] = "OMVS.PRV.FS3";
    char new_filesystemname[45] = "OMVS.PRV.FS4";

    struct parmstruct myparmstruct;

    FS_ID2 *idop = &(myparmstruct.fsid_old);
    FS_ID2 *idnp = &(myparmstruct.fsid_new);

    myparmstruct.myparms.opcode = FSOP_RENAME_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(idop,0,sizeof(FS_ID2)); /* Ensure reserved fields are 0 */
    memset(idnp,0,sizeof(FS_ID2)); /* Ensure reserved fields are 0 */

    memcpy(&myparmstruct.fsid_old.fsid_eye, FSID_EYE, 4);
    myparmstruct.fsid_old.fsid_len = sizeof(FS_ID2);
    myparmstruct.fsid_old.fsid_ver = FSID_VER_2;
    strcpy(myparmstruct.fsid_old.fsid_name,old_filesystemname);

    memcpy(&myparmstruct.fsid_new.fsid_eye, FSID_EYE, 4);
    myparmstruct.fsid_new.fsid_len = sizeof(FS_ID2);
    myparmstruct.fsid_new.fsid_ver = FSID_VER_2;
    strcpy(myparmstruct.fsid_new.fsid_name,new_filesystemname);

    BPX1PCT("ZFS ",

bad text
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 renaming file system %s to %s\n",old_filesystemname, new_filesystemname);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else    /* Return from rename file system was successful */
{
    printf("File system %s was renamed to %s successfully\n",old_filesystemname, new_filesystemname);
}
return 0;
}
Set Auditfid

Purpose
The Set Auditfid subcommand is an aggregate operation that sets the current value of the auditfid.

Format
syscall_parmlist
opcode 149 AGOP_SETAUDITFID_PARM_DATA
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
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
EBUSY auditfid could not be set
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
The aggregate whose auditfid is to be changed must be attached. 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

Restrictions
The aggregate cannot be attached as read-only. The aggregate cannot be quiesced. The aggregate cannot be in the process of being cloned, clone deleted, or moved by zFS.

Examples
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
#include <stdio.h>
Set Auditfid

```c
#define ZFSCALL_AGGR 0x40000005
#define AGOP_SETAUDITFID_PARMDATA 149 /* Set or reset 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 */
    #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"; /* aggregate name to set auditfid */

    struct parmstruct myparmstruct;
    AGGR_ID *idp = &(myparmstruct.aggr_id);

    myparmstruct.myparms.opcode = AGOP_SETAUDITFID_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = 0; /* 0=set new auditfid if current auditfid is 0 */
    /* 1=set new auditfid regardless of current value (force) */
    /* 2=set new auditfid to 0 (pre-z/OS V1R9) */
    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);

    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)
```
Set Auditfid

{
    printf("Error setting auditfid for aggregate \%s\n", aggrname);
    printf("BPXRV = \%d BPXRC = \%d BPXRS = \%x\n", bpxrv, bpxrc, bpxrs);
    return bpxrc;
}

else /* Return from set auditfid was successful */
{
    printf("Aggregate \%s set auditfid successfully\n", aggrname);
}

return 0;
}
Set Config Option

Purpose
The 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 particular configuration setting from input specified as a character string.

The following Format and Example use the CFGOP_ADM_THREADS subcommand. The other set subcommands (see "Configuration commands" on page 179) operate in a similar manner. That is, each of them set the configuration setting from the character string in the co_string field.

Format
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
Set Config Option subcommands are used to set the current value of a particular configuration option. Each subcommand sets one configuration from a character string.

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 */
    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;
```

Chapter 14. zFS application programming interfaces 271
myparmstruct.myparms.parms[5] = 0;
myparmstruct.myparms.parms[6] = 0;

memset(coptr,0,sizeof(CFG_OPTION));
coptr->co_eye=CFG_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 ",
    ZFSCALL_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 -adm_threads, 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;
Set File System Quota

Purpose

The Set File System Quota subcommand call is a file system operation that sets the quota for the file system.

You can use an FS_ID or an FS_ID2 as input.

Format

syscall_parmlist

opcode 141 FSOP_SETQUOTA_PARMDATA

parms[0] offset to FS_ID or FS_ID2

parms[1] 7000 quota

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 short sizeof(FS_ID)

fsid_ver char 1

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_reserved char[32] 0

fsid_reserved2 char[2] 0

FS_ID2 or FS_ID

fsid_eye char[4] "FSID"

fsid_len short 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

Return_code

EBUSY Aggregate containing file system is quiesced
EINTR ZFS is shutting down
EINVAL Invalid parameter list
EMVSERR Internal error using an osi service
ENOENT Aggregate is not attached
EPERM Permission denied to perform request
EROFS Aggregate is attached as read only

Reason_code

0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
Set File System Quota

Usage
The aggregate containing the file system to have its quota set must be attached. A quota can be
decreased if the quota usage has not exceeded the new quota.

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
List File System Status

Restrictions
You cannot set the quota of a backup file system. The aggregate containing the file system to have its
quota set cannot be quiesced or attached as read-only. The minimum value for quota is 128 (for 128 K
bytes).

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
Example 1 - Using FS_ID

```c
#include <stdio.h>
#define ZFSCALL_FILESYS 0x40000004
#define FSOP_SETQUOTA_PARM_DATA 141

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 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_FSYSTYPE+1]; /* Name, null terminated */
    char fsid_reserved[32]; /* Reserved for the future */
    char fsid_reserved2[2]; /* Reserved for the future */
} fs_id_t;
```

274  z/OS V1R11.0 Distributed File Service zFS Administration
Set File System Quota

} 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.PRVS.FS3";
    int quota;

    struct parmstruct myparmstruct;
    FS_ID *idp = &(myparmstruct.fsid);
    quota = 7000;

    myparmstruct.myparms.opcode = FSOP_SETQUOTA_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = quota;
    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_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 setting quota of %d for file system %s\n",quota, filesystemname);
        printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    }
    else /* Return from set quota of file system was successful */
    {
        printf("File system %s had its quota set to %d successfully\n",filesystemname, quota);
    }
    return 0;
}

Example 2 - Using FS_ID2

#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_FILESYS  0x40000004
#define FSOP_SETQUOTA_PARMDATA  141
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_mtnam[ZFS_MAX_FSYSNAME+1]; /* Mount name, null terminated */
    char fsid_reserved[49]; /* 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";
    int quota;

    parmstruct myparmstruct;
    FS_ID *idp = &(myparmstruct.fsid);
    quota = 7000;

    myparmstruct.myparms.opcode = FSOP_SETQUOTA_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);

    ...
myparmstruct.myparms parms[1] = quota;
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_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 setting quota of %d for file system %s
", quota, filesystemname);
    printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
    return bpxrc;
}
else /* Return from set quota of file system was successful */
{
    printf("File system %s had its quota set to %d successfully\n", filesystemname, quota);
}
return 0;
}
Statistics Directory Cache Information

Purpose
The statistics directory cache information subcommand call is a performance statistics operation that returns directory cache counters.

Format
syscall_parmlist

<table>
<thead>
<tr>
<th>opcode</th>
<th>249</th>
<th>STATOP_DIR_CACHE</th>
</tr>
</thead>
<tbody>
<tr>
<td>parms[0]</td>
<td>offset to STAT_API</td>
<td></td>
</tr>
<tr>
<td>parms[1]</td>
<td>offset to output buffer</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>

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_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_res1 | int | reserved |
| ad_reserved | 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 |

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
It is used to determine the numbers of requests, hits and discards from the directory cache.
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_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 */
    #define DS_EYE "ADIR"
    short ad_size; /* Size of output structure */
    char ad_version; /* Version of stats */
    #define DS_VER_INITIAL 1 /* First version of log 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_reserved2[10]; /* Reserved for future use */
} API_DIR_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 */
```

Chapter 14. zFS application programming interfaces 279
Statistics Directory Cache Information

```c
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_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(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 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));
    memset(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"); */
```

280  z/OS V1R11.0 Distributed File Service zFS Administration
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, BPXRV = %d BPXRC = %d BPXRS = %x\n",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("%10d %9d %10d %10d %3d.%1.1d%% %10d\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, myparmstruct.mystats.ad_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 iobyaggr Information

Purpose
This is information about the number of reads and writes and the number of bytes transferred for each aggregate.

Format
syscall_parmlist

opcode 244 STATOP_IOBYAGGR
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

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

total_number_waits_for_io unsigned long total number of waits for I/O

average_wait_time_for_io unsigned long average wait time (whole number)

average_wait_time_for_io_decimal unsigned long average wait time (decimal part)

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_reads unsigned long count of reads for this aggregate

temp_read_bytes unsigned long bytes read for this aggregate (in kilobytes)

temp_writes unsigned long count of writes for this aggregate

temp_write_bytes unsigned long bytes written for this aggregate (in kilobytes)

allocation_dsname char[84] data set name of aggregate

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
It is used to determine the numbers of I/Os and the amount of data transferred on an aggregate basis.

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

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 */
    u_long posix_usecs; /* microseconds */
    int pad1;
} 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*/
    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 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;
    unsigned long average_wait_time_for_io_decimal;
} 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 bpxrv;
    int bpxrc;
    int bpxrs;
    int i;
    IO_REPORT2_GRAND_TOTALS *stgt;
    IO_REPORT2 *str2;
    char *stsy;
    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) + 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_IOBYAGGR;
    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 iobyaggr 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;
    stgt = (IO_REPORT2_GRAND_TOTALS *)((char *)stapptr + sizeof(IO_REPORT2_GRAND_TOTALS));
    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 */
    /* strcpy(stsy,"DCEIMGVQ"); */

    BPX1PCT("ZFS ",
    ZFSCALL_STATS,   /* 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 */
  }
  printf(" zFS I/O by Currently Attached Aggregate\n");
  printf("\n");
  printf("DASD PAV\n");
  printf("VOLSER IOs Mode Reads K bytes Writes K bytes Dataset Name\n");

Chapter 14. zFS application programming interfaces  285
Statistics iobyaggr Information

printf("------ --- ---- ---------- ---------- ---------- ---------- ------------\n");
for( i = 0 ; i < stgt->io_count ; i++, str2++)
{
    printf("%6.6s %3d %s %10d %10d %10d %10d %-44.44s\n", 
           str2->volser, 
           str2->pavios, 
           str2->read_ind, 
           str2->temp_reads, 
           str2->temp_read_bytes, 
           str2->temp_writes, 
           str2->temp_write_bytes, 
           str2->allocation_dsname);
}
printf("%6d %10d %10d %10d %10d %-44.44s\n", 
        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: %10d\n", stgt->total_number_waits_for_io);
printf("Average I/O wait time: %9d.%3.3d (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_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);
}
free(myp);
}
else /* 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 */
{
    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 */
{
    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",bpivr, bpirc, bpirs);
}
return 0;
Statistics iobydasd Information

Purpose

This is information about the number of reads and writes and the number of bytes transferred for each DASD volume.

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 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_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 (decimal part)

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 that 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 (decimal part)
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

It is used to determine the numbers of I/Os and the amount of data transferred on a DASD basis.
Privilege Required
None.

Related Services
Statistics iobyaggr Information
Statistics icounts Information

Restrictions
None.

Examples

```c
#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 pad1;
} 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 */
/*******************************************************************************/

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;
};
```
Statistics iobydasd Information

```c
} STAT_API;

typedef struct api_iobydasd_hdr
{
    int number_of_lines;
    int pad;
    hyper grand_total_waits;
    int avg_wait_time_whole;
    int avg_wait_time_decimal;
} 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 waits;
    int avg_wait_whole;
    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 mysize, 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 ",
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) + sizeof(STAT_API) + buflen +
                  sizeof(myparmstruct.systemname);
        myp = (struct parmstruct *) malloc ((long) mypsize);
        memset(myp, 0, mypsize);

        printf("Need buffer size of %ld, for a total of %ld

", 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(STAT_API));
        stdd = (API_IOBYDASD_DATA *)((char *)stdh + sizeof(API_IOBYDASD_HDR));
        stsy = (char *)((char *)stdd + sizeof(API_IOBYDASD_HDR) + sizeof(STAT_API) + buflen);

    /* This next field should only be set if parms[2] is non-zero */
    /* strcpy(stsy,"DCEIMGVQ"); */
    }
Statistics iobydasd Information

```c
printf("VOLSER I/Os Reads K bytes Writes K bytes Waits Average Wait\n");
printf("------ --- ---------- ---------- ---------- ---------- ---------- ------------");
for (i = 0; i < stdh->number_of_lines ; i++, stdd++)
{
    printf("%6.6s %3d %10d %10d %10d %10d %10d %6d.%3.3d",
        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: %d,,%d\n",
    stdh->grand_total_waits.high,stdh->grand_total_waits.low);
printf("Average I/O wait time: %9d.%3.3d (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 */
    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);
}
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);
    }
}
```
Statistics iobyasd Information

```c
}
}
return 0;
}
```
Statistics iocounts Information

Purpose
This is 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

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]

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
It is used to determine the numbers of I/Os zFS has issued for various circumstances. The number of API_IO_BY_CIRC items can be 18 (for compatibility with previous releases) or 19.
Privilege Required

None.

Related Services

Statistics iobyaggr Information
Statistics iobydasd 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_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 epoch */
    u_long posix_time_low; /* low order 32 bits since epoch */
    u_long posix_usecs; /* microseconds */
    int pad1;
} 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_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_IO_BY_TYPE_t
{
    unsigned long number_of_lines;
```
Statistics iocounts Information

unsigned long count;
unsigned long waits;
unsigned long cancels; /* Successful cancels of IO */
unsigned long merges; /* Successful merges of IO */
char reserved1[6];
char description[51];
char padded[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 reserved1[6];
  char description[51];
  char padded[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];
  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_IOCOUTNS;
  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) */
  /* + (TOTALTYPES * 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));

  Statistics iocounts Information
  296
  z/OS V1R11.0 Distributed File Service zFS Administration
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
",stiotptr->number_of_lines);
        return 1;
    }
    if( stiocptr->number_of_lines != TOTAL_CIRC )
    {
        printf("Unexpected number of IO Circumstances, %d instead of TOTAL_CIRC
",stiocptr->number_of_lines);
        return 2;
    }
    printf(" I/O Summary By Type
");
    printf(" -------------------
");
    printf("Count Waits Cancels Merges Type
");
    printf("---------- ---------- ---------- ---------- ----------
");
    for( i=0; i<TOTAL_TYPES; i++ )
    {
        printf("%10d %10d %10d %10d %s
",
               stiotptr->count, stiotptr->waits,
               stiotptr->cancels, stiotptr->merges,
               stiotptr->description);
        stiotptr = stiotptr + 1;
    }
    printf(" I/O Summary By Circumstance
");
    printf(" ---------------------------
");
    printf("Count Waits Cancels Merges Circumstance
");
    printf("---------- ---------- ---------- ---------- ------------
");
    for( i=0; i<TOTAL_CIRC; i++ )
    {
        printf("%10d %10d %10d %10d %s
",
               stiocptr->count, stiocptr->waits,
               stiocptr->cancels, stiocptr->merges,
               stiocptr->description);
        stiocptr = stiocptr +1;
        printf("\n");
    }
    if (0==ctime_r((time_t *)&stapptr->reset_time_infoposix_time_low, buf))
    {
        printf("Could not get timestamp.
");
    }
    else
    /* Insert the microseconds into the displayable time value */
Statistics iocounts Information

```c
    strncpy(&(buf[27]),&(buf[20]),6);
    sprintf(&(buf[20]),"%06d",stapptr-&gt;reset_time_info.posix_usecs);
    buf[26]=' ';
    buf[19]='.';
    printf("Last Reset Time: %s",buf);
```
Statistics Kernel Information

Purpose
The statistics kernel information subcommand call is a performance statistics operation that returns kernel counters.

Format
syscall_parmlist

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>246 STATOP_KNPFS</td>
</tr>
<tr>
<td>parms[0]</td>
<td>offset to STAT_API</td>
</tr>
<tr>
<td>parms[1]</td>
<td>offset to output buffer</td>
</tr>
<tr>
<td>parms[2]</td>
<td>offset to system name (optional)</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>

STAT_API

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sa_eye</td>
<td>char[4] &quot;STAP&quot;</td>
</tr>
<tr>
<td>sa_len</td>
<td>int length of buffer that follows STAT_API</td>
</tr>
<tr>
<td>sa_ver</td>
<td>int 1</td>
</tr>
<tr>
<td>sa_flags</td>
<td>char[1] 0x00</td>
</tr>
<tr>
<td>sa_reset</td>
<td>0x80 Reset statistics</td>
</tr>
<tr>
<td>sa_fill</td>
<td>char[3] 0</td>
</tr>
<tr>
<td>sa_reserve</td>
<td>int[4] 0</td>
</tr>
<tr>
<td>posix_time_high</td>
<td>unsigned long high order 32 bits since epoch</td>
</tr>
<tr>
<td>posix_time_low</td>
<td>unsigned long low order 32 bits since epoch</td>
</tr>
<tr>
<td>posix_useconds</td>
<td>unsigned long microseconds</td>
</tr>
<tr>
<td>pad1</td>
<td>int</td>
</tr>
</tbody>
</table>

KERNEL_CALL_STATS

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kc_eye</td>
<td>char[8] &quot;xxxxADIR&quot;</td>
</tr>
<tr>
<td>kc_version</td>
<td>short version</td>
</tr>
<tr>
<td>kc_size</td>
<td>short size of output</td>
</tr>
<tr>
<td>pad1</td>
<td>int reserved</td>
</tr>
</tbody>
</table>

KERNEL_LINE[40]

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kl_operation_name</td>
<td>char[27] operation name string</td>
</tr>
<tr>
<td>kl_valid</td>
<td>char operation entry is valid (0x01)</td>
</tr>
<tr>
<td>kl_count</td>
<td>unsigned long count of operations</td>
</tr>
<tr>
<td>kl_time_reserved</td>
<td>int always zero</td>
</tr>
<tr>
<td>kl_time</td>
<td>int average time for operation</td>
</tr>
<tr>
<td>kl_reserved</td>
<td>int[6] reserved</td>
</tr>
<tr>
<td>kc_totalops</td>
<td>unsigned long grand total operations</td>
</tr>
<tr>
<td>pad2</td>
<td>int reserved</td>
</tr>
<tr>
<td>kc_totaltime_reserved</td>
<td>int reserved</td>
</tr>
<tr>
<td>kc_totaltime</td>
<td>int grand total wait time</td>
</tr>
<tr>
<td>kc_valid_slots</td>
<td>int number of slots in above array that actually contains data</td>
</tr>
<tr>
<td>kc_reserved</td>
<td>int[10] reserved</td>
</tr>
<tr>
<td>pad3</td>
<td>int reserved</td>
</tr>
</tbody>
</table>

systemname   | char[9]                                         |

Return_value  | 0 if request is successful, -1 if it is not successful |

Return_code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINTR</td>
<td>zFS is shutting down</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Invalid parameter list</td>
</tr>
<tr>
<td>EMVSERR</td>
<td>Internal error occurred</td>
</tr>
<tr>
<td>E2BIG</td>
<td>Information too big for buffer supplied</td>
</tr>
</tbody>
</table>

Reason_code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xEFnnxxxx</td>
<td>See z/OS Distributed File Service Messages and Codes</td>
</tr>
</tbody>
</table>
Statistics Kernel Information

**Usage**
It is used to determine the numbers of kernel operations and average time for the operation.

**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 <stdio.h>
#include <stdlib.h>
#define ZFSCALL_STATS 0x40000007
#define STATOP_KNPFS 246 /* 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 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*/
} kernel_call_stats_t;
```
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 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; /* this buffer area follows this struct*/
    int sa_flags; /* the version number currently always 1*/
    #define SA_VER_INITIAL 0x01
    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 1;
    STAT_API *stapptr = &(myparmstruct.myapi);
    KERNEL_CALL_STATS *stkcptr = &(myparmstruct.mystats);
    char buf[33];

    myparmstruct.myparms.opcode = STATOP_KNPFS;
    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[2] = 0;
Statistics Kernel Information

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(KERN_API);
/* This next field should only be set if parms[2] is non-zero */
/* strcpy(myparmstruct.systemname,"DCEIMGVQ"); */
BPX1PCT("ZFS ",
     ZFSCALL_STATS, /* Perf statistics operation */
    offsetof(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",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else
{
    printf("%34s
","zFS Kernel PFS Calls");
    printf("%34s
","--------------------");
    printf("%34s
","Operation Count Avg Time
");
    printf("--------- ---------- ----------
");
    i=0;
    while (myparmstruct.mystats.OUTPUT[i].kl_valid == 1)
    {
        printf("%13s %10u %9d.%3.3d
",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("--------- ---------- ----------
");
    printf("%10u %9d.%3.3d\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

The statistics locking information subcommand call is a performance statistics operation that returns 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_ver int 1
sa_flags char 0x80 for reset; 0x00 otherwise
sa_fill char[3] 0
sa_reserve int[4] 0

STAT_LOCKING

reserved1 int
stlk_untimed_sleeps unsigned long number of untimed sleep
stlk_timed_sleeps unsigned long number of timed sleep
stlk_wakesups unsigned long number of wake ups
stlk_total_wait_for_locks unsigned long total waits for locks
stlk_average_lock_wait_time double average lock wait time
stlk_avg_lock_wait_time_whole int average lock wait time in msecs left of the decimal part
stlk_avg_lock_wait_time_decimal int average lock wait time in msecs decimal portion
stlk_total_monitored_sleeps unsigned long total monitored sleeps
stlk_average_monitored_sleep_time double average monitored sleep time
stlk_avg_mon_sleep_time_whole int average monitored sleep time in msecs left of the decimal portion
stlk_avg_mon_sleep_time_decimal int average monitored sleep time in msecs decimal portion
stlk_total_contentions unsigned long total lock contention of all kinds
stlk_reserved_space char[48] reserved for future use
stlk_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 percentage < 1
  description char[84] Description of the lock
  sleeppcount 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

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
Usage
This function is used to retrieve locking information.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required
None.

Related Services
Statistics Storage Information
Statistics User Cache Information

Restrictions
None.

Examples

```c
#include <stdlib.h>
#include <stdio.h>
#define ZFSCALL_STATS 0x40000007
#define STATOP_LOCKING 240 /* Performance API queries */

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 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*/
    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 */
    char description[84]; /* Description of the thread*/
```
Statistics Locking Information

```c
#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*/
#define SA_RESET 0x80
char sa_flags; /* flags field must be x00 or x80, x80 means reset statistics*/
#define SA_RESET INITIAL 0x80
char sa_fill[3]; /* spare bytes */
int sa_reserve[4]; /* Reserved */
struct reset_time reset_time_info;
} STAT_API;

typedef struct stat_locking_t
{
  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 */
  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 */
  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;

struct parmstruct
{
```
```
Statistics Locking Information

```c
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 *stapptr = &(myparmstruct.myapi);
    STAT_LOCKING *stlkptr = &(myparmstruct.mystats);
    char buf[33];

    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_LOCKING);

    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;

    memcpy(stapptr->sa_eye,SA_EYE,4);
    stapptr->sa_ver=SA_VER_INITIAL;
    stapptr->sa_len=(int) sizeof(STAT_LOCKING);

    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("Untimed sleeps: %10d  Timed Sleeps: %10d  Wakeups: %10d\n",mystats.stlk_untimed_sleeps,mystats.stlk_timed_sleeps,mystats.stlk_wakeups);
        printf("\n");
        printf("Total waits for locks: %10d\n",mystats.stlk_waits);
```
myparmstruct.mystats.stlk_total_wait_for_locks);

printf("Average lock wait time: %9d.%3.3d (msecs)\n",
     myparmstruct.mystats.stlk_avg_lock_wait_time_whole,
     myparmstruct.mystats.stlk_avg_lock_wait_time_decimal);

printf("\n");

printf("Total monitored sleeps: %10d\n",
     myparmstruct.mystats.stlk_total_monitored_sleeps);

printf("Average monitored sleep time: %9d.%3.3d (msecs)\n",
     myparmstruct.mystats.stlk_avg_mon_sleep_time_whole,
     myparmstruct.mystats.stlk_avg_mon_sleep_time_decimal);

printf("\n");

printf(" Top %d Most Highly Contended Locks\n",
     MAX_LOCKS);

for (i=0 ;i<MAX_LOCKS;i++ )
{
    printf("%10d %10d %10d %3d.%1.1d%% %.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: %10d\n",
     myparmstruct.mystats.stlk_total_contentions);

printf("\n");

printf(" Top %d Most Common Thread Sleeps\n",
     MAX_SLEEPS);

for( i = 0; i < MAX_SLEEPS;i++ )
{
    printf(" %10d %3d.%1.1d%% %.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);
}

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);
    printf(&buf[20]),"%06d",stapptr->reset_time_info posix_usecسد);;
    buf[26]=';
    buf[19]='.;'
    printf("Last Reset Time: %s",buf);
}

return 0;
Statistics Log Cache Information

Purpose
The statistics log cache information subcommand call is a performance statistics operation that returns log cache counters.

Format
```
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  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_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_writtenPages_reserved int      reserved
    al_writtenPages int      number of log buffer pages written to disk
    al_fullWaits_reserved int      reserved
    al_fullWaits int      number of times new log buffer requires wait on prior log pages
    al_nbsWaits_reserved 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
EMVSERR Internal error occurred
E2BIG Information too big for buffer supplied

Reason_code
0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
Usage
It is used to determine the numbers of requests, hits and waits on the log buffer cache.

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_LOG_CACHE 247 /* Performance API queries */
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_LOG_STATS_t 
{
  char al_eye[4]; /* Eye catcher = ALOG */
  #define LS_EYE "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 */
  hyper al_nbsWaits; /* Number of time new log buffer requires wait on new block user IO */
  int al_reserved3[10]; /* Reserved for future use */
} API_LOG_STATS;
Statistics Log Cache Information

/* 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];     /* length of the buffer to put data into*/
    int sa_len;        /* 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_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.myapi);
    /* 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;

    /* 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 log cache statistics of a different system than this one */

    myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) + sizeof(API_LOG_STATS); */

    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    /* 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 */
    /*********************************************************************/

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_LOG_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 log cache, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
return bpxrc;
}
else
{
 printf("%52s\n","Log File Caching Statistics");
 printf("\n");
 printf("Buffers (K bytes) Requests Hits Ratio Written \n");
 printf("---------- --------- ---------- ---------- ------ ----------\n");

 temp_ratio = (myparmstruct.mystats.al_lookups.low == 0) ? 0.0 : 
 ((double)myparmstruct.mystats.al_hits.low)/myparmstruct.mystats.al_lookups.low;
 temp_ratio *= 100.0;
 CONVERT_RATIO_TO_INTS(temp_ratio, whole, decimal);
 decimal = decimal / 100; /* Just want tenths */
 printf("%10d %9d %10d %10d %3d.%1.1d%% %10d\n",
 myparmstruct.mystats.al_buffers.low,
 myparmstruct.mystats.al_buffers.low * myparmstruct.mystats.al_buffsize,
 myparmstruct.mystats.al_lookups.low, myparmstruct.mystats.al_hits.low,
 whole, decimal, myparmstruct.mystats.al_writtenPages.low);
 printf("\n");
 printf("New buffer: log full waits %10d NBS IO waits %10d\n",
 myparmstruct.mystats.al_fullWaits.low, myparmstruct.mystats.al_nbsWaits.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);
  printf(&buf[20],"%06d",stapptr->reset_time_info.posix_usecs);
  buf[26]=';
  buf[19]='.
  printf("Last Reset Time: %s",buf);
 }

 return 0;
}
Statistics Metadata Cache Information

Purpose

The statistics metadata cache information subcommand call is a performance statistics operation that returns metadata cache counters.

Format

```
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
  buffsize 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

BACK_STATS
  buffers hyper number of buffers in the cache
  buffsize 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

  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
It is used to determine the numbers of requests, hits and discards from the directory cache.

Privilege Required
None.

Related Services
  Statistics Vnode Cache Information
  Statistics Metadata Cache Information

Restrictions
None.

Examples
#define ZFSCALL_STATS 0x40000007
#define STATOP_META_CACHE 248 /* Metadata cache (and back cache) stats */

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 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 */
} PRIMARY_STATS_t;
Statistics Metadata Cache Information

```c
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 {
    int 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 for future use */
} 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 */
    int pad1;
} RESET_TIME;

/*****************************************************************************/
/* The following structure is the api query control block */
/*****************************************************************************/
#define SA_EYE "STAP"
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*/
    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];
}
```
```c
void main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    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[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"); */
    BPXIPCT("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 meta cache, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    }

    /* Primary cache */

    printf("%52s\n","Metadata Caching Statistics");
    printf("\n");
    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;
    CONVERT_RATIO_TO_INTS(temp_ratio, whole, decimal);
    decimal = decimal / 100; /* Just want tenths */

    printf("%10d %9d %10d %10d %3d.%1.1d%% %10d\n",)
```

Chapter 14. zFS application programming interfaces 315
Statistics Metadata Cache Information

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.hits.low,
whole, decimal, myparmstruct.mystats.am_primary.updates.low);

printf(" \n");

/* Backing cache */
printf("%56s\n","Metadata Backing Caching Statistics");
printf(" \n");
printf("Buffers (K bytes) Requests Hits Hits Ratio Discards\n");
printf("---------- --------- ---------- ---------- ------ ----------\n");

double 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("%10d %9d %10d %10d %3d.%1.1d%% %10d
",

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);
    printf("%46d",stapptr->reset_time_info.posix_usecs);
    buf[26] = ' ';
    buf[19] = '.';
    printf("Last Reset Time: %s",buf);
}

return 0;
}
Statistics Storage Information

Purpose
The statistics storage information subcommand call is a performance statistics operation that returns storage information.

Format
syscall_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_ver int 1
  sa_flags char 0x80 for reset; 0x00 otherwise
  sa_fill char[3] 0
  sa_reserve int[4] 0

API_STOR_STATS
  reserved 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*/
  ss_comp_pieces int /* The number of pieces allocated*/
  ss_comp_allocations int /* the number of storage allocations requests*/
  ss_comp_frees int /* the number of storage frees done by this*/
  ss_comp_description char[84] /* the component description */
  ss_number_of_detail_lines int /* the number of detail lines following this*/

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
Statistics Storage Information

Usage
This function is used to retrieve storage information. 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
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_STATS 0x40000007
#define STATOP_STORAGE 241 /* 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*/
    /* 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 comp_line
{
    int ss_comp_bytes_allocated; /* The number of bytes allocated by this component */
    int ss_comp_pieces; /* The number of pieces allocated*/
    int ss_comp_allocations; /* the number of storage allocations requests done by this component */
    int ss_comp_frees; /* the number of storage frees done by this component */
    char ss_comp_description[84]; /* the component description */
    int ss_number_of_detail_lines; /* the number of detail lines following this component line */
    /* before the next component line or end of buffer */
} COMP_LINE;

typedef struct detail_line
{
    int ss_detail_bytes_allocated; /*number of bytes allocated*/
    int ss_detail_pieces; /*number of pieces allocated*/
    int ss_detail_allocations; /*number of allocation requests*/
    int ss_detail_frees; /*number of free requests*/
    char ss_detail_description[84]; /*description */
} DETAIL_LINE;

typedef struct api_stor_stats
{
    int reserved1;
    unsigned long ss_total_bytes_allocated; /* Total bytes allocated*/
    unsigned long ss_total_pieces_allocated; /* Total pieces allocated*/
    unsigned long ss_total_allocation_requests; /* Total allocation requests*/
    unsigned long ss_total_free_requests; /* Total free requests*/
    unsigned long ss_number_of_comp_lines; /* Total number of components lines in buffer*/
    char ss_reserved_space[48]; /* reserved for future use */
}/*********************************************************************/
/**The returned data can contain comp_lines and detail_lines ******/
/**The first line is a component line ******/
/**The number of component lines returned is in this structure ******/
/* Each component line is followed by zero or more detail lines */
/* The comp_line struct indicates how many detail lines follow */
/***************************************************************************/
} API_STOR_STATS;

struct parmstruct
{
    syscall_parmlist myparms;
    STAT_API myapi;
    /* output buffer API_STOR_STATS + COMP_LINES and DETAIL_LINES */
    char systemname[9];
} myparmstruct;

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    int i,j;
    COMP_LINE *stcl;
    DETAIL_LINE *stdl;
    char *stsy;
    char buf[33];

    struct parmstruct *myp = &myparmstruct;
    int mypsize, buflen;
    API_STOR_STATS *stst;
    STAT_API *stapptr = &(myparmstruct.myapi);
myparmstruct.myparms.opcode = STATOP_STORAGE;
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 storage statistics of a different system than this one */

/* This next field should only be set if parms[2] is non-zero */

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 */
        mysize = sizeof(syscall_parmlist) + sizeof(STAT_API) + buflen +
                sizeof(myparmstruct.systemname);
        myp = (struct parmstruct *) malloc ((long) mysize);
        memset(myp, 0, mysize);

        printf("Need buffer size of %d, for a total of %d\n", buflen, mysize);

        myp->myparms.opcode = STATOP_STORAGE;
        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 storage statistics of a different system than this one */

        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;
        stst = (API_STOR_STATS *)((char *)myp + sizeof(syscall_parmlist) + sizeof(STAT_API));
        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"); */
BPX1PCT("ZFS ",
ZFS_CALL_STATS, /* Aggregate operation */
myssize, /* 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 )
{
  printf(" zFS Primary Address Space Storage Usage\n");
  printf("-------------------------------------------------------------\n");
  printf("\n");
  printf("Total Bytes Allocated: %d (%dK) (%dM)\n",
stst->ss_total_bytes_allocated,
stst->ss_total_bytes_allocated/1024,
stst->ss_total_bytes_allocated/(1024*1024));
  printf("Total Pieces Allocated: %d\n",
stst->ss_total_pieces_allocated);
  printf("Total Allocation Requests: %d\n",
stst->ss_total_allocation_requests);
  printf("Total Free Requests: %d, %d\n",
stst->ss_total_free_requests, stst->ss_number_of_comp_lines);
  stcl = (COMP_LINE *)((char *)stst + sizeof(API_STOR_STATS));
  for( i = 0 ; i < stst->ss_number_of_comp_lines ; i++ )
  {
    printf("\n");
    printf(" Storage Usage By Component\n");
    printf("-------------------------------------------------------------\n");
    printf("Bytes No. of No. of \n"
      "Allocated Pieces Allocs Frees Component\n");
    printf("---------- ------ ------ ------ ---------\n");
    printf("\n");
    printf("%10d %6d %6d %6d %s\n",
stcl->ss_comp_bytes_allocated,
stcl->ss_comp_pieces,
stcl->ss_comp_allocations,
stcl->ss_comp_frees,
stcl->ss_comp_description);
    stdl = (DETAIL_LINE *)((char *)stcl + sizeof(COMP_LINE));
    for( j = 0 ; j < stcl->ss_number_of_detail_lines ; j++, stdl++ )
    {
      if( j == 0 )
      {
        printf("\n");
        printf(" Storage Details by Component\n");
        printf("-------------------------------------------------------------\n");
        printf("\n");
      }
      printf("%10d %6d %6d %6d %s\n",
        stdl->ss_detail_bytes_allocated,
        stdl->ss_detail_pieces,
        stdl->ss_detail_allocations,
        stdl->ss_detail_frees,
        stdl->ss_detail_description);
} stcl = (COMP_LINE *)stdl;
}
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);
}
free(myp);
}
else /* storage stats failed with large enough buffer */
{
    printf("Error on storage stats with large enough buffer\n");
    printf("Error querying storage stats, BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv,bpxrc,bpxrs);
    free(myp);
    return bpxrc;
}
else /* error was not E2BIG */
{
    printf("Error on storage stats 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 Transaction Cache Information

Purpose

The statistics transaction cache information subcommand call is a performance statistics operation that returns transaction cache counters.

Format

syscall_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_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_usec  unsigned long microseconds
  pad1  int

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

systemname  char[9]

Return_value  0 if request is successful, -1 if it is not successful

Return_code

  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

It is used to determine the numbers of transactions in the transaction cache.
Statistics Transaction Cache Information

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_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*/
    int sa_reserve[4]; /* Reserved */
    #define SA_RESET 0x80
    char 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)
{
    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;
    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);
    sttcptr->sa_len=0;

    BPX1PCT("ZFS ",
            ZFSCALL_STATS,  /* Perf statistics operation */
            sizeof(myparmstruct), /* Length of Argument */
            (char *) &myparmstruct,  /* Pointer to Argument */
            &bpxrv, /* Pointer to Return_value */
            Statistics Transaction Cache Information
            Chapter 14. zFS application programming interfaces  325
Statistics Transaction Cache Information

```c
#include <stdio.h>
#include <time.h>

void printStats() {
    if (bpxrv < 0) {
        printf("Error querying tran cache, BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
        return bpxrc;
    }

    printf("Transaction Cache Statistics\n");
    printf("Trans started: %8d Lookups on Tran: %8d EC Merges: %8d\n", myparmstruct.mystats.sttr_started,
            myparmstruct.mystats.sttr_lookups,
            myparmstruct.mystats.sttr_ec_merges);
    printf("Allocated Trans: %8d (Act= %7d, Pend= %7d, Comp= %7d, Free= %7d)\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]), 6);
        sprintf(&(buf[20]), "%06d", stapptr->reset_time_info.posix_usecs);
        buf[26] = '\n';
        buf[19] = '.';
        printf("Last Reset Time: %s\", buf);
    }

    return 0;
}
```

Statistics Transaction Cache Information

326
z/OS V1R11.0 Distributed File Service zFS Administration
Statistics User Cache Information

Purpose
The statistics user cache information subcommand call is a performance statistics operation that returns user cache information.

Format
```
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[3]   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_readasyncs u_long
  vm_writes  u_long
  vm_getattrs u_long
  vm_flushes u_long
  vm_scheduled_deletes u_long
  vm_reads_faulted u_long
  vm Writes_faulted u_long
  vm_read_ios u_long
  vm_scheduledWrites u_long
  vm_error_writes u_long
  vm_reclaim_writes 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 u_long
  vm_reclaim_steal 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]
```

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

Reason code
0xEFxxnnnn See z/OS Distributed File Service Messages and Codes
Statistics User Cache Information

Usage
This function is used to retrieve cache information.

Reserved fields and undefined flags must be set to binary zeros.

Privilege Required
None.

Related Services
Statistics Locking Information
Statistics Storage Information

Restrictions
None.

Examples

```c
#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_alloc_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 epoc */
    u_long posix_time_low; /* low order 32 bits since epoch */
    u_long posix_usecs; /* microseconds */
    int pad1;
} RESET_TIME;

typedef struct stat_api_t
{
    #define SA_EYE "STAP"
} stat_api_t;
```

Statistics User Cache Information

```c
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;

/* 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;
    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 */
    u_long vm_reserved[10]; /*reserved for future use*/
} 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_vmsSegTable_cachesize; /* Number of segments */
  DS_ENTRY stuc_ds_entry[32]; /* Array of dataspace 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[33];

  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;

  /* 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); */
  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_USER_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 user cache stats, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
    return bpxrc;
}
else
{
    printf(" User File (VM) Caching System Statistics\n");
    printf(" ----------------------------------------\n");
    printf("\n");
    for( i = 0 ; i <=REMOTE ; i++ )
    {
        if( i == 0 )
        {
            printf(" Direct Statistics\n");
            printf(" -----------------\n");
            printf("\n");
        }
        else
        {
            printf(" Client Statistics\n");
            printf(" -----------------\n");
            printf("\n");
        }
        printf("External Requests:\n");
        printf("------------------\n");
        printf("%-9s %10u %-9s %10u %-9s %10u\n", "Reads",myparmstruct.mystats.stuc[i].vm_reads,
            "Fsyncs",myparmstruct.mystats.stuc[i].vm_fsyncs,
            "Schedules",myparmstruct.mystats.stuc[i].vm_schedules);
        printf("%-9s %10u %-9s %10u %-9s %10u\n", "Writes",myparmstruct.mystats.stuc[i].vm_writes,
            "Setattrs",myparmstruct.mystats.stuc[i].vm_setattrs,
            "Unmaps",myparmstruct.mystats.stuc[i].vm_unmaps);
        printf("%-9s %10u %-9s %10u %-9s %10u\n", "Asy Reads",myparmstruct.mystats.stuc[i].vm_readasyncs,
            "Getattrs",myparmstruct.mystats.stuc[i].vm_getattrs,
            "Flushes",myparmstruct.mystats.stuc[i].vm_flushes);
        printf("\n");
    }
    printf("File System Reads:\n");
    printf("------------------\n");
    ratio1 = ratio2 = ratio3 = ratio4 = 0.0;
    if( myparmstruct.mystats.stuc[i].vm_reads > 0 )
    {
        ratio1 = 100 * (((double)myparmstruct.mystats.stuc[i].vm_reads_faulted)
            / (double)myparmstruct.mystats.stuc[i].vm_reads));
    }
    if( myparmstruct.mystats.stuc[i].vm_writes > 0 )
    {
        ratio2 = 100 * (((double)myparmstruct.mystats.stuc[i].vm_writes_faulted)
            / (double)myparmstruct.mystats.stuc[i].vm_writes));
    }
    if( myparmstruct.mystats.stuc[i].vm_read_ios > 0 )
    {
        ratio3 = 100 * (((double)myparmstruct.mystats.stuc[i].vm_read_ios)
            / (double)myparmstruct.mystats.stuc[i].vm_reads));
    }
    printf("%-14s %10u (%s Ratio %.2f%%)\n", "Reads Faulted",myparmstruct.mystats.stuc[i].vm_reads_faulted,
            "Fault",ratio1);
    printf("%-14s %10u (%s Ratio %.2f%%)\n", "Writes Faulted",myparmstruct.mystats.stuc[i].vm_writes_faulted,
            "Fault",ratio2);
    printf("%-14s %10u (%s Ratio %.2f%%)\n", "Read Waits",myparmstruct.mystats.stuc[i].vm_read_ios,
            "Wait",ratio3);
    printf("\n");
}
printf("File System Writes:\n");
printf("-------------------\n");
printf("%-19s %10u %-13s %10u\n", "Scheduled Writes", myparmstruct.mystats.stuc[i].vmScheduled_writes, "Sync Waits", myparmstruct.mystats.stuc[i].vmSync_waits);
printf("%-19s %10u %-13s %10u\n", "Error Writes", myparmstruct.mystats.stuc[i].vmError_writes, "Error Waits", myparmstruct.mystats.stuc[i].vmError_waits);
printf("%-19s %10u %-13s %10u\n", "Page Reclaim Writes", myparmstruct.mystats.stuc[i].vmReclaim_writes, "Reclaim Waits", myparmstruct.mystats.stuc[i].vmReclaim_waits);
if( myparmstruct.mystats.stuc[i].vm_writes>0)
{
    ratio4 = 100 * (((double)myparmstruct.mystats.stuc[i].vmWrite_waits) / ((double)myparmstruct.mystats.stuc[i].vm_writes));
}
printf("%-19s %10u (Wait Ratio %.2f%%)\n", "Write Waits", myparmstruct.mystats.stuc[i].vmWrite_waits, ratio4);
printf("\n");
printf("Page Management (Segment Size = (%dK Local %dK Remote) ) (Page Size = %dK)\n", myparmstruct.mystats.stuc_seg_size_loc, myparmstruct.mystats.stuc_seg_size_rmt, myparmstruct.mystats.stuc_page_size);
printf("-------------------------------------------------------------------------\n");
printf("Total Pages %10u Free %10u\n", myparmstruct.mystats.stuc_cache_pages,myparmstruct.mystats.stuc_total_free);
printf("Segments %10u\n", myparmstruct.mystats.stuc_vmSegTable_cachesize);
printf("Steal Invocations %10u Waits for Reclaim %11u\n", myparmstruct.mystats.stuc[0].vm_reclaim_steal, myparmstruct.mystats.stuc[0].vm_waits_for_reclaim);
printf("\n");
printf("Number of dataspaces used: %5d ",myparmstruct.mystats.stuc_dataspaces);
printf("Pages per dataspace: %11d\n",myparmstruct.mystats.stuc_pages_per_ds);
printf("\n");
printf("Dataspace Allocated Free\n");
printf("Name Segments Pages\n");
printf("-------- ---------- ----------\n");
for( i=0;i<myparmstruct.mystats.stuc_dataspaces ; i++ )
{
    printf("%-8s %10u %10u\n", myparmstruct.mystats.stuc_ds_entry[i].ds_name, myparmstruct.mystats.stuc_ds_entry[i].ds_alloc_segs, myparmstruct.mystats.stuc_ds_entry[i].ds_free_pages);
}
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 Vnode Cache Information

Purpose
The statistics vnode cache information subcommand call is a performance statistics operation that returns vnode cache counters.

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_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

STAT_VNODE_CACHE
VNM_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)
Allocates hyper allocates
Deletes hyper deletes
VnodeStructSize hyper base vnode structure size
ExtendedVnodes hyper number of extended vnodes
extensionSize hyper size of vnode extension
USSHeldVnodes hyper number of held vnodes
USSHeldVnodesHi 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

ZFSVNODEOPCOUNCTS
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]
Statistics Vnode Cache Information

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
This function is used to determine the numbers of requests, hits and discards from the vnode cache.

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_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;

typedef struct hyper {
    unsigned long high; /* unsigned long reserved */
    unsigned long low;
} hyper;

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 */
}
Statistics Vnode Cache Information

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;
    hyper RatioDecimal;
    hyper Allocates;
    hyper Deletes;
    hyper VnodeStructSize;
    hyper ExtendedVnodes;
    hyper extensionSize; /* (minimum) in bytes */
    hyper USHeldVnodes;
    hyper USHeldVnodesHi;
    hyper OpenVnodes;
    hyper OpenVnodesHi;
    hyper OpenVnodesReuse;
    long reserved1[3];
    int pad1;
    hyper reserved2[10];
} 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];
    ZFSVNODEOPCOUNTS zFSOpCounts[50];
} EFS_STATS_API_STRUCT;

typedef struct stat_vnode_cache_t
{
    VNM_STATS_API_STRUCT vnm_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_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*/
}
typedef int SA_RESET 0x80;

char sa_fill[3]; /* spare bytes */
int sa_reserve[4]; /* Reserved */

struct reset_time {
    char *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_ver = 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, "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 vnode cache, BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
        return bpxrc;
    }
}

Statistics Vnode Cache Information
```

```

Statistics Vnode Cache Information

```c

```
Statistics Vnode Cache Information

```c
}
return 0;
}
```
Unquiesce Aggregate

Purpose
The Unquiesce Aggregate subcommand call is an aggregate operation that unquiesces a multi-file system aggregate on a system. This allows activity on the aggregate and all its file systems to resume.

Format
syscall_parmlist
  opcode 133 AGOP_UNQUIESCE_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.PRV.AGGR001.LDS0001"
  aid_reserved char[33] 0

Return_value 0 if request is successful, -1 if it is not successful

Return_code
  EINTR ZFS is shutting down
  EMWSERR 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
The unquiesce call must supply the quiesce handle that was returned by the quiesce call. The aggregate is typically quiesced prior to 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
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);
Unquiesce Aggregate

```c
#include <stdio.h>
#include <stdlib.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_UNQUIESCE_PARMDATA 133

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_Ten;
    /* 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";
    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[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,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);

    BPXIPCT("ZFS ",
        ZFSCALL_AGGR,  /* Aggregate operation */
        sizeof(myparmstruct),  /* Length of Argument */
    );
}
```

340  z/OS V1R11.0 Distributed File Service zFS Administration
(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
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. In order to do this, you must:

- replace the BPX1PCT with BPX4PCT
- replace the #pragma linkage(BPX1PCT, OS) statement with #pragma linkage(BPX4PCT, OS64_NOSTACK)
- change all the "long" declares to "int"
- Ensure there are appropriate includes for function calls
- Ensure all functions requiring 64-bit parameters are passing 64-bit numbers (for example, ctime_r).

The remaining code is, or can remain unchanged. The following is an example with these changes.
Statistics iocounts information

Examples

```c
#include <stdio.h>
#include <time.h>

#define ZFSCALL_STATS 0x40000007
#define STATOP_IOCOUNCTS 243 /* Performance API queries */

#define TOTAL_TYPES 3
#define TOTAL_CIRC 18

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_usecs; /* microseconds */
    int pad1;
} 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_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 */
} 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
```
Statistics iocounts information

```c
{  
    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;

strtolower(**********
/* The following structures are used to represent cfgop queries */
/* for iocounts */
strtolower(**********

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 1;
    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_IOCOUNSTS;
    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, BXPVRV = %d BXRRC = %d BXRRS = %x\n",bpxrv,bpxrc,bpxrs);
        return bpxrc;
    } 
    else
    {
        if( stiotptr->number_of_lines != TOTAL_TYPES )
```

Appendix A. Running the zFS pfsctl APIs in 64-bit mode  345
Statistics iocounts information

    
    
    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(" -------------------\n");
printf(" Count Waits Cancels Merges Type\n");
printf("---------- ---------- ---------- ---------- ----------\n");
for( i=0; i<TOTAL_TYPES; i++ )
{
    printf("\%10d \%10d \%10d \%10d \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(" Count Waits Cancels Merges Circumstance\n");
printf("---------- ---------- ---------- ---------- ------------\n");
for( i=0; i<TOTAL_CIRC; i++ )
{
    printf("\%10d \%10d \%10d \%10d \s\n", stiocptr->count, stiocptr->waits, stiocptr->cancels, stiocptr->merges, stiocptr->description);
    stiocptr = stiocptr + 1;
    printf("\n");
}

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",stapptr->reset_time_info.posix_usecs);
    buf[26] = '\';
    buf[19] = '.';
    printf("Last Reset Time: \%s\n",buf);
}
return 0;
}
Appendix B. Accessibility

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](#) 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.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at:

Notices

This information was developed for products and services offered in the USA.

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
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

© Copyright IBM Corp. 2001, 2009
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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

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.

Programming Interface Information

This z/OS Distributed File Service System z File System Administration primarily documents information that is NOT intended to be used as Programming Interfaces of the Distributed File Service.

This z/OS Distributed File Service System z File System 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

The following terms are trademarks of the International Business Machines Corporation in the United States, or other countries, or both:

• BookManager
• DFS
• DFSMSdss
• IBM
- IBMLink
- Library Reader
- Lotus Notes
- MVS
- RACF
- Resource Link
- RMF
- System z
- System z9
- z/OS
- zSeries
- z/VM

UNIX is a registered trademark of The Open Group in the United States and other countries.

Microsoft and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.
Glossary

This glossary includes terms and definitions for Distributed File Service zSeries 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 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 to the owning file system and returning the response to the requester through XCF communications.

global resource serialization (GRS). 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 (GRS) to serialize access to shared resources such as data sets on shared direct access storage device (DASD) volumes.

GRS. See global resource serialization

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. In zFS, pertaining to a physical file system that handles file requests for mounted file systems by remotely shipping function 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.** In zFS, pertaining to a physical file system that handles file requests for mounted file systems locally instead of shipping function requests through z/OS UNIX.

**version file system.** See root file system.

**zFS.** See zSeries file system.

**zFS aggregate.** A Virtual Storage Access Method Linear Data Set (VSAM LDS) that contains zFS file systems.

**zSeries 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.
Special characters
# (pound sign)  ix

A
access control lists (ACL)  3
accessibility  347
ACL (access control lists)  3
address space  17
determining usage  70
aggregate
  back up  43
corruption  75
disabled  84
restore  44
aggregate attach subcommand  182
aggregate movement  3
aggregate operations  179
aggregate state
determining  81
aggregates  40
comparing  54
disabled  84
  compatibility mode aggregate  84
  multi-file system aggregate  84
full  53
growing
  multi-file system  53
  multi-file system  49, 54
valid characters in name  102
allocation
  blocked  25, 58
  fragmented  25, 58
  inline  25, 58
anode  107
API (Application Programming Interface)  3
pfsctl  177
Application Programming Interface (API)  3
pfsctl  177
ASID
determining  81
attach aggregate subcommand  182
auditid  90
auditids
  converting  90
auditid
  contents  91
  enabling  90
  example  89
  overview  89
tool  89

B
back up (continued)
  how to  43
  restore  44
  using DFSMSdss logical dump  43
backslash ix
backup file system  24
batch job  114
blocked file allocation  25, 58
bpxmtext  101
  zFS  101

C
cache
debugging  70
log file  59
metadata  58
transaction  58
user file  59
vnode  58
cache report
  VM  63
  vnode  65
cache size
  IOEFSPRM  57
  storage shortage  83
  total  57
catch-up mount  31
cron  24
cron file system subcommand  186
cloning  4
cron file system  24
cron message  82
cron status
determining  81
cosexistence
  IOEFSPRM and IOEPRMxx  167
command
  zfsadm attach  121
command suites
  zfsadm  113
commands
  bpxmtext  101
  ioeagfmt  102
  ioeagslv  105
  modify  74
  modify zfs process  96
  mount  20, 109
  setomvs reset  99
  TSO/E
    mount  52
  z/OS system  95
  zfsadm aggrinfo  21, 53, 54, 117
  zfsadm apropos  120
  zfsadm attach  50, 166
  zfsadm clone  24, 124
  zfsadm clonesys  24, 126
  zfsadm config  128

© Copyright IBM Corp. 2001, 2009
commands (continued)
  zfsadm configquery 131
  zfsadm create 51, 134
  zfsadm define 137
  zfsadm delete 139
  zfsadm detach 141
  zfsadm format 143
  zfsadm grow 21, 53, 145
  zfsadm help 147
  zfsadm lsaggr 148
  zfsadm lsfs 149
  zfsadm lsquota 51, 151, 153
  zfsadm rename 158
  zfsadm setauditfid 160
  zfsadm setquota 51, 53, 161
  zfsadm unquiesce 163
comparing
  aggregates 54
compatibility mode aggregate 54
  adding a volume 22
deleting 23
  disabled 84
renaming 23
size 28
compatibility mode file system 19
  maximum size 28
  minimum size 28
mounting 20
concept
  file system ownership 5
  z/OS UNIX 5
  zFS 5
configuration operations 179
configuring 14
considerations
  sysplex 31
conventions
  this document ix
corruption
  file system 75
create file system in aggregate 191
create file system subcommand 191
creating
  compatibility mode file system 19
  multi-file system aggregates 49
  zFS file system 19
CTKC
  report 73

definitions (continued)
  local mount 6
  non-sysplex aware 6
  read-only file system 6
  read-write file system 6
  shared file system environment 6
  sysplex 6
  sysplex-aware 6
  z/OS UNIX file system owner 5
  zFS aggregate 6
  zFS file system 7
  zFS file system name 7
  zFS file system owner 5
  zFS physical file system 7
delay
delay of mount 23, 24
delays
  understanding 80
delete file system subcommand 201
detach aggregate subcommand 206
determining
  zFS active 18
  DFSMSdss logical dump
    using for back up 43
directory
  size 28
    determining 29
directory space
  how to reclaim 29
disability 347
disabled aggregates
  compatibility mode aggregate 84
  multi-file system aggregate 84
  non-sysplex aware 85
  sysplex-aware 84
dump
  obtaining 77
  overview 76

E
examples
  clone file system 187
  cloning 81
  create file system 192
  create multi-file system aggregate 50
  creating compatibility mode file system 19
  define aggregate 198
  delete file system 202
  detach aggregate 207
  format aggregate 210
  grow aggregate 214
  ioeafmt 103
  ioeagslv command 107
  IOEFSPRM sample file 174
  list aggregate status 217
  list aggregate status (version 2) 221
  list attached aggregate names 225
  list attached aggregate names (version 2) 229
  list file system names 233, 237

D
data sets
  IOEFSPRM 166
debugging 74
  storage shortage 83
define aggregate subcommand 197
definitions
  anode 107
  backup file system 4
  catch-up mount 4
  file system ownership 5
examples (continued)
  list systems 249
  logical restore 45
  modify zfs process 98
  query config option 254
  quiesce aggregate 259
  rename file system 263
  replace 45
  restore 44
  set config option 267, 271
  set file system quota 274
  setomvs reset 99
  statistics directory cache information 279
  statistics ibyagg information 283
  statistics ibydasd information 289
  statistics icounts information 295
  statistics kernel information 300
  statistics locking information 304
  statistics log cache information 309
  statistics metadata cache information 313
  statistics storage information 318
  statistics transaction cache information 324
  statistics user cache information 328
  statistics vnode cache information 334
  unquiesce aggregate 339
  zfsadm aggregate restore 44
  zfs back up 44
  zfsadm aggrinfo command 118
  zfsadm apropos command 120
  zfsadm attach command 123
  zfsadm clone command 125
  zfsadm clonesys command 127
  zfsadm config command 130
  zfsadm configquery command 133
  zfsadm create command 135
  zfsadm define command 138
  zfsadm delete command 140
  zfsadm grow command 146
  zfsadm help command 147
  zfsadm lsaggr command 148
  zfsadm lsfs -long 81
  zfsadm lsfs command 150
  zfsadm lsquota command 152, 153
  zfsadm quiesce command 156
  zfsadm rename command 159
  zfsadm setauditfid command 160
  zfsadm setquota command 162
  zfsadm unquiesce command 163

file allocation
  blocked 25, 58
  fragmented 25, 58
  inline 25, 58

file system
  active 18
  backup 24
  cloning 24
  corruption 75
  determining owner 35
  dynamic movement 38
  full 53
  maximum size 28
  minimum size 28
  ownership 36
  read-only
    sysplex-aware 31
  read-write 24
  read/write
    non-sysplex aware 32
    sysplex-aware 33
  status 18
  valid characters in name 134
  z/OS UNIX owner 36

file system operations 179
file system owner
  z/OS UNIX 34
  zFS 34

file system ownership 5
files
  IOEFSPRM 166
fixed storage 60
format aggregate subcommand 209
fragmented file allocation 25, 58

G
grow aggregate subcommand 213
growing
  multi-file system aggregate 53

H
hand detector
  using 79
hang
  detection 79
  steps for resolving 80
hangs
  understanding 80
HFS
  auditid 89

I
I/O
  statistics 72
I/O balancing 60
IBM HTTP Server
  Fast Response Cache Accelerator 7
  restriction 8
in a shared file system environment
  multi-file system 40
inline file allocation 25, 58
installation
  post 11
installing 14
intermediate archive file 47
IOBYDASD
  report 72
ioegfmt command 102
  example 103
ioegfmt utility 19
ioeagslv 75
  disabled aggregate 84
ioeagslv command 105
  example 107
IOEFSPRM 166
  example 174
  sharing 40
  total cache size 57
IOEFSPRM and IOEPRMxx
  coexistence 167
IOEPRMxx 166
  example 166

J
JES 14, 15

K
keyboard 347

L
large directory 29
list aggregate status (version 2) subcommand 220
list aggregate status subcommand 216
list attached aggregate name subcommand 224
list attached aggregate names (version 2)
  subcommand 228
list file system names subcommand 232, 236
list file system status subcommand 240
list systems subcommand 249
local mount 6, 31
LOCK report 69
log file cache 59
log files 59

M
mainframe
  education x
man
  command 101
man pages
  enabling 101
  example of command 101
  for zFS 101
managing
  processes 17
  zFS file system 19
messages 168, 169, 170
  overview 77
metadata 24
metadata cache 58
migrating
  from HFS to zFS 47
  using the z/OS HFS to zFS migration tool 47
modify command 74
modify zfs process command 96
  examples 98
mount command 20, 52, 109
mount wait 23, 24
mounting
  compatibility mode file system 20
  same mode 85
multi-file system aggregates 40, 49, 54
  creating 49
  disabled 84
  growing 53
multilevel security 3

N
namespace 78
namespace correction 78
namespace validation 78
NBS (New Block Security) 121, 122
New Block Security (NBS) 121, 122
NLS 168, 169, 170
non-sysplex aware 6, 32
  zFS 36
NOREADAHEAD option 59
Notices 349

O
objects
  maximum number 28
options
  NOREADAHEAD 59
zFS PFS 167
overview 3

P
path entry 111
pax command 47
performance 3, 57
  number of file names 28
PFS
  state 18
PFS (physical file system) 7, 167
PFS Calls
  client 62
  owner 62
plfctl
  aggregate operations 179
  configuration operations 179
pfsclt  (continued)
file system operations  179
query operations  179
physical file system (PFS)  7
post installation processing  11
pound sign (#) ix

Q
query config option subcommand  253
query operations  179
QUERY.KN report  61, 62
QUERY.STOR  70
quiesce aggregate subcommand  258
quota  51

R
RACF
  authority  15
  commands  14
read-only
  file system  31
read-only file system  6
read-write file system  6, 24
read/write
  file system  32
reason codes
  using bpxmtext  101
rename file system subcommand  261
replace  45
restart  3
restore
  from back up  44, 45
restriction
  IBM HTTP Server  8
  SMB  7
root
  large directory  29

S
salvager
  when to use  75
salvager utility  75
same mode  85
security label  3
service level
  how to determine  78
set auditfid subcommand  267
set config option subcommand  270
set file system quota subcommand  273
setomvs reset command  99
  examples  99
shared file system  31
  overview  31
shared file system environment  6
sharing zfs data between systems  27, 54
shortcut keys  347
SMB
  non-sysplex aware  38

SMB  (continued)
  ownership  38
  restriction  7
SMF record
  auditid  89
statistics directory cache information subcommand  278
statistics iboyaggr information subcommand  282
statistics iboyasid information subcommand  288
statistics iocounts information subcommand  294
statistics kernel information subcommand  299
statistics linking information subcommand  303
statistics log cache information subcommand  308
statistics metadata cache information subcommand  312
statistics storage information subcommand  317
statistics transaction cache information subcommand  323
statistics user cache information subcommand  327
steps
  creating
    compatibility mode file system  19
  installing  14
STKM
  report  72
STOR  70
storage
  debugging  70
storage shortage
  causes  83
storing files
  blocked  58
  fragmented  58
  inline  58
subcommands
  aggregate attach  182
  attach aggregate  182
  clone file system  186
  create file system  191
  define aggregate  197
  delete file system  201
  detach aggregate  206
  format aggregate  209
  grow aggregate  213
  list aggregate status  216
  list aggregate status (version 2)  220
  list attached aggregate name  224
  list attached aggregate names (version 2)  228
  list file system names  232, 236
  list file system status  240
  list systems  249
  query config option  253
  quiesce aggregate  258
  rename file system  261
  set auditfid  267
  set config option  270
  set file system quota  273
statistics directory cache information  278
statistics iboyaggr information  282
statistics iboyasid information  288
statistics iocounts information  294
subcommands (continued)
  statistics kernel information 299
  statistics locking information 303
  statistics log cache information 308
  statistics metadata cache information 312
  statistics storage information 317
  statistics transaction cache information 323
  statistics user cache information 327
  statistics vnode cache information 333
  unquiesce aggregate 339
SVI call 73
sysplex 6
  considerations 31
  z/OS UNIX consideration 40
sysplex-aware 6
  file system 31
  mixed with non-sysplex aware 37
  overview 31

T
tasks
  hang, resolving
  steps 80
token manager
  statistics 72
total cache size 57
trace
  overview 74
  steps 74
transaction cache 58
TSO/E commands
  mount 52

U
unclone 139
unquiesce
  operator command 98
  zfsadm command 163
unquiesce aggregate subcommand 339
user file cache 59
using PARMLIB (IOEPRMxx) 166

V
valid characters in aggregate name 102
valid characters in file system name 134
VM cache report 63
vnode cache 58
VSAM Linear Data Set (LDS) 19, 49

Z
z/OS
  system commands 95
    modify zfs process 96
    setomvs reset 99
  UNIX commands
    pax 47
z/OS Basic Skills information center x
z/OS UNIX file system owner 5, 34
z/OS UNIX owner 36
zFS (System z File System)
  managing processes 17
zFS (zSeries File System) 3
  back up 43
zFS address space 17
zFS aggregate 6
zFS disk space allocation 25
zFS file system name 7
zFS file system owner 5, 34
zFS file systems 7
  creating 19
  managing 19
zFS physical file system (PFS) 7
  options 167
zFS reason codes
  displaying 101
zfsadm aggrinfo command 21, 53, 54, 117
  example 118
zfsadm apropos command 120
  example 120
zfsadm attach 121
  format 121
  options 121
  usage 122
zfsadm attach command 50, 166
  example 123
  privilege 122
zfsadm clone command 24, 124
  example 125
zfsadm clonesys command 24, 126
  example 127
zfsadm command suite
  command syntax 113
  introduction 113
zfsadm commands 113
zfsadm config command 128
  example 130
zfsadm configquery command 131
  example 133
zfsadm create command 51, 134
  example 135
zfsadm define command 137
  example 138
zfsadm delete command 139
  example 140
zfsadm detach command 141
zfsadm format command 143
zfsadm grow command 21, 53, 145
  example 146
zfsadm help command 147
  example 147
zfsadm lsaggr command 148
  example 148
zfsadm lsfss command 149
  example 150
zfsadm lsquota command 51, 151, 153
  example 152, 153
zfsadm quiesce command  
example  156
zfsadm rename command  158  
example  159
zfsadm setauditfid command  160  
example  160
zfsadm setquota command  51, 53, 161  
example  162
zfsadm unquiesce command  
example  163
zSeries File System (zFS)  
features  3
overview  3
We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:
• Send your comments to the address on the reverse side of this form.
• Send your comments via e-mail to: mhvcfs@us.ibm.com

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address
Program Number: 5694-A01

Printed in USA