z/OS

IBM

Distributed File Service zFS Administration

Version 2 Release 2

SC23-6887-01

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

Contents

Figures

Tables

About this document

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

How this document is organized

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

- Part 1, "zFS administration guide," on page 1 provides guidance information for the z/OS File System (zFS).
- Part 2, "zFS administration reference," on page 113 provides reference information about z/OS File System (zFS), which includes z/OS system commands, zFS commands, and zFS data sets.

Conventions used in this document

This document uses the following typographic conventions:

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

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

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

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

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

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

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

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

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

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

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

z/OS information

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

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

To find the complete z/OS library, go to IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

References to DFS information

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

Information about z/OS File System messages and codes is found in *z/OS Distributed File Service Messages and Codes*.

How to send your comments to IBM

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

Use one of the following methods to send your comments:
1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html).

Include the following information:
- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
  z/OS V2R2 Distributed File Service zFS Administration
  SC23-6887-01
- The topic and page number that is related to your comment.
- The text of your comment.

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

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

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:
- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at z/OS Support Portal (http://www-947.ibm.com/systems/support/z/zos/).

Summary of changes

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

Summary of changes for z/OS Version 2 Release 2 (V2R2)

The following changes are made in z/OS Version 2 Release 2 (V2R2).

New

- "What's new or changed for zFS in z/OS V2R2" on page 8 was added.
- zFS can be run in the OMVS address space, which is used by z/OS UNIX. See "zFS running in the z/OS UNIX address space" on page 19.
- You can display detailed information about the zFS file system. See "Usage notes for displaying file system information" on page 121 and "Examples of displaying file system information" on page 122.
- MODIFY ZFS PROCESS has a new parameter, **`fsinfo`**, which displays detailed information about zFS file systems. Usage notes and examples for displaying file system information were also added. See "MODIFY ZFS PROCESS" on page 116.
- The **`zfsadm config`** and **`zfsadm configquery`** commands have a new option, `-modify_cmd_threads`. It specifies the current number of threads that are defined to handle zFS modify commands. See "zfsadm config" on page 165 and"zfsadm configquery" on page 169.
- The **`zfsadm fsinfo`** command displays detailed information about zFS file systems. See "zfsadm fsinfo" on page 191.
- New reports are available that can be printed with the **`zfsadm query`** command using the keywords `-stkm`, `-ctkc`, and `-svi`. This information is also available in new application programming interfaces for Client Token Caching Component, Server Token Manager, and Statistics from the Server Vnode Interface. For more information about the keywords, see "zfsadm query" on page 209.
- The IOEFSPRM configuration options file has new options.
  - The `modify_cmd_threads` option controls the number of modify commands that are running simultaneously.
  - The `user_running_hangdump` option specifies whether a hang dump should be taken for a user task that has been hanging for approximately 5 minutes.
  - The `quiesceinfo_message_delay` option specifies the minimum number of seconds to delay issuing the IOEZ00830E message. See "Processing options for IOEFSPRM and IOEPRMxx" on page 222.
- The pfsctl (BPX1PCT) application programming interface was updated to include a new command, ZFSCALL_FSINFO. Two subcommands (**`query modify_cmd_threads`** and **`set modify_cmd_threads`**) were added to ZFSCALL_CONFIG. See Table 17 on page 235.
- These application programming interfaces (APIs) were added:
  - "List Detailed File System Information" on page 276. It lists detailed file or directory information.
  - "Statistics Server Token Management Information" on page 405. It returns statistics for the server token manager.

- "Statistics Sysplex Client Operations Information" on page 423. It returns information about the number of local operations that required the sending of a message to another system.
- "Statistics Sysplex Owner Operations Information" on page 430. It returns information about the number of calls that are processed on the local system as a result of a message that was sent from another system.

Changed

- Information about updating required APARs was updated for V2R2. See "Applying required APARs for z/OS V2R2" on page 16.
- zFS caches can now be obtained in virtual storage above the 2 GB bar (64-bit storage). As a result, much larger caches can be used to increase zFS performance. zFS performance can further be increased because it can be run in the OMVS address space, which is used by z/OS UNIX. See "zFS running in the z/OS UNIX address space" on page 19.
- Clarification was added about the statistics that zFS supplies for SMF type 30 records. See "Support for type 30 SMF record" on page 45.
- "Performing a backup of zFS " was renamed to "Copying or performing a backup of a zFS" and a warning was added. See Chapter 6, "Copying or performing a backup of a zFS," on page 61.
- To handle the case of the source file system having symbolic links (or names) longer than 100 characters, the **pax** examples in "Using an intermediate archive file" on page 66 and "Without using an intermediate archive file" on page 66 were updated to include the `-o saveext` option.
- Chapter 8, "Performance and debugging," on page 67 was updated.
- The QUERY,KN report was updated because the statistics report now allows for larger counter values to be displayed. These displays will use a suffix indicating the multiplier that is to be used for the displayed counter value. See "KN" on page 76.
- The LFS report was updated because large fast lookup statistics is no longer supported. See "LFS" on page 78.
- Information about thrashing was added to the STKM report. See "STKM" on page 83.
- The STOR report was updated. See "STOR" on page 85.
- The VM report was updated because client caching is no longer supported. See "VM" on page 89.
- Release updates were made to "Determining service levels" on page 97.
- Starting in V2R2, zFS uses the enhanced log and enhanced status APIs XCF communication protocol. Previously, it used the extended directory XCF communications protocol. For more information, see "Determining the XCF protocol interface level" on page 97.
- The `-client_cache_size` and `-tran_cache_size` keywords for the **zfsadm config** and **zfsadm configquery** commands are no longer supported. If they are specified, they are accepted but not used.
- Various updates were made to the IOEFSPRM configuration options file.
  - The `client_cache_size` option is now ignored because V1R12 can no longer exist in the sysplex.
  - The `tran_cache_size` option is now ignored because there is no longer a separate transaction cache.

  – The expected default value for the `meta_cache_size` and `metaback_cache_size` options were changed because the entire calculated default for the size of the metadata cache is now assigned to `meta_cache_size`.
  – The upper end of the expected value for `token_cache_size` was changed from 2621440 to 20 million.
  – The upper end of the expected value for `meta_cache_size` was changed from 1024 M to 64 G.
  – The upper end of the expected value for `trace_table_size` and `xcf_trace_table_size` were changed from 2048 M to 65535 M.
  – The upper end of the expected value for `vnode_cache_size` was changed from 500000 to 10 million.

  See “IOEFSPRM” on page 220.
- The APIs in Chapter 13, “zFS application programming interface information,” on page 233 were reformatted. One of the changes was that `long` was changed to `int` because the length of a `long` can be 4 bytes or 8 bytes, depending on compiler options.
- The pfsctl (BPX1PCT) application programming interface was updated to include a new command, ZFSCALL_FSINFO. See “pfsctl (BPX1PCT)” on page 234.
- The Statistics Log Cache Information format was changed because a new log cache facility is used in V2R2. New statistics are returned pertaining to this new logging method. See “Statistics Log Cache Information” on page 387.
- Statistics Storage Information returns information for storage above the 2 G addressing bar. See “Statistics Storage Information” on page 411
- The Statistics Transaction Cache Information is no longer used, but documentation about it was kept. See “Statistics Transaction Cache Information” on page 438.

Deleted

- The section "Transaction cache" in Chapter 8, “Performance and debugging,” on page 67 was deleted because a separate transaction cache no longer exists.
- The `flc` IOEPRMxx configuration option was deleted because it is no longer supported.
- The Delete File System API was deleted because it is no longer supported.
- The sections “zFS support for read/write non-sysplex aware mounted file system” and “zFS support for read/write sysplex aware mounted file system” were deleted because they described how zFS in V1R11 or V1R12 systems handled read/write file systems. These systems can no longer exist in the sysplex with V2R2.
- The section “Disabled aggregates when there are no z/OS V1R13 or later systems” was deleted because all systems must now be at z/OS V1R13 or later.
- The description of the IOEFSPRM configuration file option `client_cache_size` was deleted because V1R12 can no longer exist in the sysplex.
- The description of the IOEFSPRM configuration file option `tran_cache_size` was deleted because the new zFS aggregate metadata logging method does not require a transaction cache.
- Information about large fast lookup statistics was deleted because it is no longer supported.
- Information about large FLC processing was deleted because it is no longer supported.

Summary of changes for z/OS Version 2 Release 1

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

- *z/OS Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*
- *z/OS Introduction and Release Guide*

Part 1. zFS administration guide

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

Chapter 1. z/OS File System overview

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

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

zFS can run sysplex-aware for read/write mounted file systems and for read-only mounted file systems. For more information, see “Specifying zFS file systems as sysplex-aware” on page 16, “Terminology and concepts” on page 4, and Chapter 5, “Using zFS in a shared file system environment,” on page 51.

Beginning with z/OS V1R13, zFS has enhanced its sysplex-aware support. For many file operations, zFS can now directly access zFS read/write mounted file systems in a shared file system environment from zFS client systems. In z/OS V1R13 and later releases, when zFS runs in a shared file system environment, zFS always runs sysplex-aware on a file system basis (sysplex=filesys). See “zFS-enhanced sysplex-aware support” on page 53 for more information.

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

Note:

1. Beginning with z/OS V2R1, zFS no longer supports multi-file system aggregates. If you have data that is stored in zFS multi-file system aggregates, copy that data from the zFS multi-file system aggregate file systems into zFS compatibility mode aggregates. Because zFS multi-file system aggregates cannot be mounted in z/OS V2R1, you must copy the data from any file systems that are contained in multi-file system aggregates into zFS compatibility mode file systems using a non-shared file system environment on a system that is running a release prior to z/OS V2R1.
2. Beginning with z/OS V2R1, zFS no longer supports clones. If you have read-only clone (.bak) file systems, you should delete them using the **`zfsadm delete`** command on a system that is running a release prior to z/OS V2R2.
3. Beginning with z/OS V2R2, zFS will only allow aggregates that contain exactly one file system in it to be attached.

Features

zFS provides many features and benefits, which are described in the following sections:

Performance
: zFS provides significant performance gains in many customer environments. zFS provides additional performance improvements when running sysplex-aware in a shared file system environment.

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

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

Terminology and concepts

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

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

A zFS data set can also be attached (by issuing the **`zfsadm attach`** command) without mounting it. Beginning in z/OS V2R2, only zFS data sets that contain exactly one file system are allowed to be attached. However, there are many restrictions in this case. For example, the zFS data set would not be available to z/OS UNIX applications because it was not mounted. In a shared file system environment, the zFS data set would be detached, not moved, if the system went down or zFS internally restarted. You might attach a zFS data set to explicitly grow it (**`zfsadm grow`**) or to determine the free space available (**`zfsadm aggrinfo`**). You must detach the zFS data set (**`zfsadm detach`**) before mounting it.

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

If the corresponding local mount is successful, z/OS UNIX does not function ship from that system to the z/OS UNIX owning system when that file system is accessed. Rather, the file request is sent directly to the local PFS. This is sometimes referred to as Client=N, as indicated by the output of the D OMVS,F operator command, or **`df -v`** shell command. If the corresponding local mount is unsuccessful (for instance, DASD is not accessible from that system), z/OS UNIX function ships requests to the z/OS UNIX owning system when that file system is accessed (message

BPXF221I might be issued). This is sometimes referred to as Client=Y, as indicated by the output of the D OMVS,F or **df -v** commands. For examples of the command output, see "Determining the file system owner" on page 54.

File system ownership

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

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

zFS file system owner

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

z/OS UNIX file system owner

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

For details about sysplex considerations and the shared file system environment, see "Determining the file system owner" on page 54 and Chapter 5, "Using zFS in a shared file system environment," on page 51.

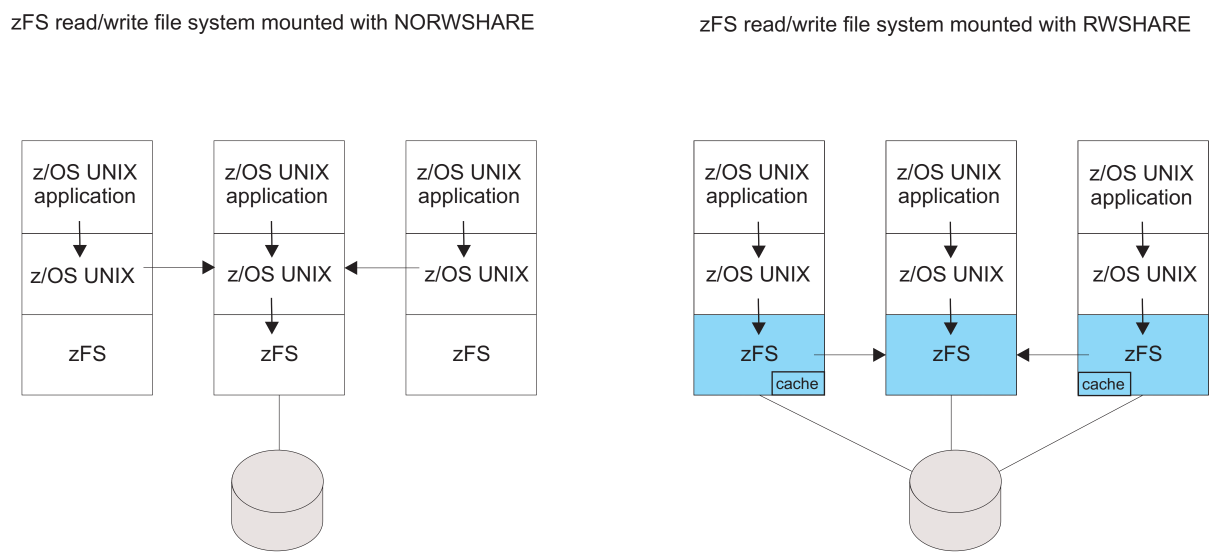
zFS read/write file system mounted with NORWSHARE
zFS read/write file system mounted with RWSHARE
z/OS UNIX application
z/OS UNIX
zFS
cache

Figure 1. z/OS UNIX and zFS file system ownership

When a file system is not sysplex-aware (that is, mounted as NORWSHARE), file requests are function-shipped by z/OS UNIX to the z/OS UNIX file system owner, and then to the PFS. When a file system is sysplex-aware (that is, mounted as RWSHARE), file requests are sent directly to the local zFS PFS and then function-shipped by zFS to the zFS file system owner, if necessary.

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

Local mount
A local mount means that z/OS UNIX issues a successful mount to the local PFS, which in this case is zFS. z/OS UNIX does this when either the file system is mounted sysplex-aware for that mode (read/write or read-only) or the system is the z/OS UNIX owner. When a file system is locally mounted on the system, z/OS UNIX does not function ship requests to the z/OS UNIX owning system. To determine if a system has a local mount, see "Determining the file system owner" on page 54.

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

OMVS address space
The address space used by z/OS UNIX, it runs a program that initializes the kernel. Starting in V2R2, zFS can be run in the OMVS address space.

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

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

Shared file system environment
The *shared file system environment* refers to a sysplex that has a BPXPRMxx specification of SYSPLEX(YES).

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

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

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

Sysplex-aware file system
A file system can be mounted sysplex-aware or non-sysplex aware. When a file system is mounted sysplex-aware, it means that the file system is locally mounted on every system (when the PFS is capable of handling a local mount on every system - that is, the PFS is running sysplex-aware) and therefore, file requests are handled by the local PFS. All read-only mounted file systems are always mounted sysplex-aware (see Figure 16 on page 52). HFS read/write mounted file systems are always mounted non-sysplex aware. This means that file requests from non z/OS UNIX owning systems are always function-shipped by z/OS UNIX to the z/OS UNIX owning system where the file system is locally mounted and the I/O is actually done.

| Beginning with z/OS V1R11, zFS read/write mounted file systems can be mounted sysplex-aware or or non-sysplex aware.

| **zFS address space**
| Because zFS can run in its own colony address space or inside the OMVS address space, which is the address space used by z/OS UNIX, any reference to the zFS address space will mean the address space in which zFS is running.

zFS aggregate

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

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

zFS file system

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

zFS Physical File System (PFS)

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

ZFS PROC

The PROC that is used to start ZFS. It is typically called ZFS. If ZFS is running in the OMVS address space, then this refers to the OMVS PROC.

What's new or changed for zFS in z/OS V2R2

In z/OS V2R2, zFS caches were moved above the 2 G addressing bar to allow for the use of very large zFS caches. These IOEFSPRM configuration variables were changed to support the following ranges of values:

Variable	Range of values
vnode_cache_size	1000 to 10000000
meta_cache_size	1 M to 64 G
token_cache_size	20480 to 20000000
trace_table_size	1 M to 65535 M
xcf_trace_table_size	1 M to 65535 M

With the zFS caches above the 2-G addressing bar, zFS can now be run inside the OMVS address space. This change yields improved performance for each file or directory operation.

The metaback cache is no longer a separate cache in a data space. It is combined with `meta_cache_size` into one single metadata cache. For simplicity and to avoid future confusion, update the IOEFSPRM configuration file to combine these two options and remove the `metaback_cache_size` setting from the file.

zFS performance counters were changed from 4 bytes to 8 bytes. This change allows for monitoring of zFS performance over longer periods of time before the counters wrap. The counters are made available via the zFS Statistics Application Programming Interfaces. This information is available in the zFS modify and **`zfsadm query`** command reports.

- New reports are available that can be printed with the **`zfsadm query`** command using the keywords `-stkm`, `-ctkc`, and `-svi`. This information is also available in new Application Programming Interfaces for Client Token Caching Component, Server Token Manager, and Statistics from the Server Vnode Interface. For more information about the keywords, see “zfsadm query” on page 209.

The **zfsadm -storage** report now contains information about storage usage above the 2 G bar.

The new **zfsadm fsinfo** command displays detailed information for one or more file systems. File systems can be specified with a specific name, or in a group by using a common prefix or common suffix. They can also be selected by specifying common attributes. Another way to obtain the detailed information is by using the new File System Information Application Programming Interface or the **modify zfs,fsinfo** command.

zFS is using a better performing method for handling the writing of records to the zFS aggregate log. The new logging method displays different statistics in the **zfsadm query -logcache** command and in the MODIFY ZFS,QUERY,LOG performance report. The Statistics Log Cache Information Application Programming Interface will also return new statistics pertaining to this new logging method.

Health checks ZOSMIGV1R13_ZFS_FILESYS and ZOSMIGREC_ZFS_RM_MULTIFS were removed, and CACHE_REMOVALS was added. For more information, see *z/OS Migration*.

For information about interface updates in V2R2, see “Summary of changes for z/OS Version 2 Release 2 (V2R2)” on page xv.

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

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

The following options are no longer supported on **zfsadm config**:
- `-fsgrow`
- `-user_cache_readahead`

The following options are no longer supported on **zfsadm configquery**:
- `-auto_attach`
- `-fsgrow`
- `-user_cache_readahead`

The following **pfsctl** subcommands are no longer supported:
- On the Aggregate command:
  - Create File System
- On the File System command:
  - Clone File System
  - Rename File System
  - Set File System Quota
- On the Config command:
  - Query auto_attach setting

  – Query fsgrow setting
  – Set fsgrow
  – Set user_cache_readahead

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

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

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

Quiesce processing for zFS file systems has been modified in z/OS V2R1. The zFS commands and zFS APIs used to quiesce and unquiesce zFS file systems are unchanged, but the way quiesce works internally and the way the quiesce status is displayed are modified.

In z/OS V2R1, the name "zSeries File System" was changed to "z/OS File System". The document *z/OS Distributed File Service zSeries File System Administration* was retitled to *z/OS Distributed File Service zFS Administration*.

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

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

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

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

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

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

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

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

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

A new configuration variable was added to IOEFSPRM: `user_running_hangdump`.

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

Chapter 2. Installing and configuring zFS

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

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

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

- Decide if you want to run zFS in its own colony address space or in the OMVS address space. For more information that you can use to help make this decision, see "zFS running in the z/OS UNIX address space" on page 19.
- Define the zFS physical file system to z/OS UNIX
- Create or update the zFS parameter data set (IOEFSPRM); see "IOEFSPRM" on page 220.
- Define zFS aggregates and file systems
- Create mount points and mount zFS file systems
- Change owner/group and set permissions on the file system root
- Optionally, add MOUNT statements in your BPXPRMxx member to cause zFS file systems to be mounted at IPL.

zFS installation and configuration steps

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

1. Install and perform postinstallation of the Distributed File Service by following the applicable instructions in *z/OS Program Directory* or in *ServerPac: Installing Your Order*. The following is a summary of the information that is contained in those documents:
   a. Ensure that the target and distribution libraries for the Distributed File Service are available.
   b. Run the `prefix.SIOESAMP(IOEISMKD)` job from UID 0 to create the symbolic links that are used by the Distributed File Service. This job reads the member `prefix.SIOESAMP(IOEMKDIR)` to delete and create the symbolic links.
   c. Ensure that the DDDEFS for the Distributed File Service are defined by running the `prefix.SIOESAMP(IOEISDDD)` job.
   d. Install the Load Library for the Distributed File Service. The Load Library (hlq.SIEALNKE) must be APF-authorized and must be in link list.
   e. Install the samples (hlq.SIOESAMP).
   f. Install the sample PROC for ZFS (hlq.SIOEPROC).
   g. One method of providing an IOEFSPRM configuration file is to define it as a data set with an IOEZPRM DD card. If zFS is to run in the OMVS address space, the IOEZPRM DD card should be placed in the OMVS PROC. If zFS

is to run in its own colony address space, create a JCL PROC for the zFS started task in SYS1.PROCLIB by copying the sample PROC from the previous step.

The DDNAME IOEZPRM identifies the optional zFS configuration file. Although this DD statement is optional, it is recommended that it be included to identify the parameter data set to be used for zFS. For now, it is suggested that this DD refer to a PDS with a member called IOEFSPRM that has a single line that begins with an asterisk (*) in column 1. Subsequent modifications can be made to the IOEFSPRM member, see "IOEFSPRM" on page 220.

As the preferred alternative to the IOEZPRM DDNAME specification, delete the IOEZPRM DDNAME and use the IOEPRMxx parmlib member. In this case, the member has the name IOEPRMxx, where you specify *xx* in the parmlib member list. See "IOEFSPRM" on page 220 for more information.

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

h. Add the following RACF® commands:

```
ADDGROUP DFSGRP SUPGROUP(SYS1) OMVS(GID(2))
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
```

The DFS user ID must have at least ALTER authority to all VSAM linear data set that contain zFS aggregates. A user ID other than DFS can be used to run the zFS started task if it is defined with the same RACF characteristics as shown for the DFS user ID. If zFS is to run in the OMVS address space, it will run under the OMVS user ID. For this reason, the OMVS user ID should be given the same RACF characteristics as the DFS user ID.

As an alternative to PERMIT ALTER authority to all VSAM linear data set that contain zFS aggregates, you can assign the zFS started task the TRUSTED attribute or you can assign the user ID of the zFS started task the OPERATIONS attribute.

2. Create a BPXPRMxx entry for zFS.

Add a FILESYSTYPE statement to your BPXPRMxx parmlib member:

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

Specifying the ASNAME(ZFS) keyword causes zFS to run in its own colony address space. To have zFS run in the OMVS address space, omit the ASNAME keyword.

```
FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSCM)
```

Recommendation: You should also specify KERNELSTACKS(ABOVE) when zFS is running in the OMVS address space. See the BPXPRMxx topic in *z/OS MVS Initialization and Tuning Reference* for more information about the KERNELSTACKS parameter.

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 including SUB=MSTR as in the following example:

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

To use the IOEPRMxx parmlib members (mentioned in step 1g on page 13), specify the *xx* values in the FILESYSTYPE statement for zFS as in the following example:

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

In this case, you must not have an IOEZPRM DD statement in your ZFS PROC. Step 4 contains an explanation as to why you should not have an IOEZPRM DD. For more information about using IOEPRMxx, see "IOEFSPRM" on page 220. S

3. Run the **dfs_cpfiles** program.

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

   To run the **dfs_cpfiles** program:
   - Logon as root (UID 0) on the local z/OS system.
   - From the z/OS UNIX shell, enter `/usr/lpp/dfs/global/scripts/dfs_cpfiles`.
4. (Optional) Create or update the zFS configuration options file (IOEPRMxx, also known as IOEFSPRM).

   The zFS configuration options file is optional. There are two methods to specify the zFS configuration options file: use IOEPRMxx in the parmlib or use an IOEZPRM DD statement in the PROC that is used to start the address space where zFS is running.
   - As the preferred alternative to the IOEZPRM DD statement, the IOEFSPRM member can be specified as a true parmlib member. In this case, the member has the name IOEPRMxx, where xx is specified in the parmlib member list. You must omit the IOEZPRM DD statement in the PROC that is used to start the address space in which zFS will run. The IOEPRMxx configuration options file can be specified with no options contained in it. Options are only required if you want to override the default zFS options. As mentioned in step 1g on page 13, it is recommended that you create an empty IOEPRMxx parmlib member. The IOEPRMxx member should only contain one line that is a comment (an asterisk (*) in column 1). See "IOEFSPRM" on page 220 for more information.
   - If you use the IOEZPRM DD statement, the PDS (organization PO) to which it points should have a record format of FB with a record length of 80. The block size can be any multiple of 80 that is appropriate for the device. A sample IOEFSPRM is provided in hlq.SIOESAMP(IOEFSPRM). IOEFSPRM is also known as IOEZS001. See "IOEFSPRM" on page 220 for a description of the IOEFSPRM options. Update the IOEZPRM DD statement in the OMVS or ZFS PROC to contain the name of the IOEFSPRM member, as shown in the following example:

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

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

     ```
     IOEZPRM DD DSN=SYS4.&SYSNAME..PARMLIB(IOEFSPRM),DISP=SHR
     ```
5. (Optional) Preallocate data sets for debugging.

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

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

   A zFS file system resides in a zFS aggregate. A zFS aggregate is a VSAM linear data set. See Chapter 4, "Creating and managing zFS file systems using compatibility mode aggregates," on page 23 for details on creating zFS file systems.

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

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

   You can create a directory with the z/OS UNIX **mkdir** command or you can use an existing directory. The TSO/E MOUNT command or the `/usr/sbin/mount` REXX exec can be used to mount the zFS file system on the directory. See Chapter 4, "Creating and managing zFS file systems using compatibility mode aggregates," on page 23 for details on mounting zFS file systems.

   Note: Steps 6 and 7 can be repeated as many times as necessary for each permanently mounted zFS file system. Only step 6 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.PRV.COMPAT.AGGR001') TYPE(ZFS) MOUNTPOINT('/etc/mountpt')
   ```

Applying required APARs for z/OS V2R2

In z/OS V2R2, in addition to the "zFS installation and configuration steps" on page 13, you must apply zFS coexistence function, using the following APAR procedure:

1. Install APAR OA46026 on all z/OSV1R13 and z/OS V2R1 systems. This APAR is a conditioning function for zFS on z/OS V1R13 and V2R1. Make APAR OA46026 active on all systems through a rolling IPL.

Specifying zFS file systems as sysplex-aware

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

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

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

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

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

- The z/OS Distributed File Service SMB server cannot export any zFS read/write file systems that are sysplex-aware.
- The Fast Response Cache Accelerator support of the IBM HTTP Server for z/OS V5.3 uses an API called register file interest (BPX1IOC using the Iocc#RegFileInt subcommand). This API cannot support zFS sysplex-aware read/write file systems, so therefore the Cache Accelerator support is not able to cache static Web pages contained in files in a zFS read/write sysplex-aware file system. Other servers that use this API can also be impacted. Generally, these are servers that cache files and must be aware of file updates from other sysplex members without having the server read the file or the file modification timestamp.
- The Policy Agent (Pagent) server, which is part of the z/OS Communications Server, cannot export any zFS read/write file systems that are sysplex-aware.

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

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

Using zFS read/write sysplex-aware file systems

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

Consider which zFS read/write file systems you might want to be sysplex-aware. Good candidates are zFS read/write file systems that are accessed from multiple systems or are mounted with AUTOMOVE and might be moved by z/OS UNIX (as a result of a shutdown or IPL) to systems that do not necessarily do the most accesses. Be aware that RWSHARE file systems use more virtual storage in the zFS
| address space than NORWSHARE file systems. Beginning in z/OS V2R2, this
| storage will be 64 bit storage (above the 2 G line). You must be careful that you do
| not use more real or auxiliary storage in the system than is needed. See the sample zFS query report “STOR” on page 85 for information about monitoring storage usage in the zFS address space. Generally, the system-specific file system (and

/dev, /etc, /tmp, /var) should be mounted as NORWSHARE and UNMOUNT because they typically are accessed only from the owning system.

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

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

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

Note: If any zFS read/write file systems were previously mounted as NORWSHARE, they will usually remain non-sysplex aware until they are un-mounted and then mounted back on the RWSHARE system. However, there are situations when the sysplex awareness might change. See "Changing zFS attributes on a mounted zFS compatibility mode file system" on page 40 for more information.

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

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

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

Alternatively, you can also issue the **f zfs,query,file** console command. As Figure 30 on page 74 shows, an "S" indicates that the zFS read/write file system is mounted sysplex aware. Because you do not have to be running in the shell, this command can be useful if a file system is under recovery or having other problems.

To change the sysplex-awareness of a mounted zFS read/write file system, you need to perform the following steps:
- Unmount the file system
- Specify the MOUNT PARM (RWSHARE to make it sysplex-aware; NORWSHARE to make it non-sysplex aware)
- Mount the file system again.

If you want to change the sysplex-awareness and you have not specified either the RWSHARE or NORWSHARE MOUNT PARM, you can change the sysplex-awareness with remount. To do so:
- Remount the file system to read-only
- Move z/OS UNIX ownership of the file system (if necessary) to a system that has `sysplex_filesys_sharemode` specified to the sharemode that you want (rwshare or norwshare)
- Remount the file system back to read/write.

zFS running in the z/OS UNIX address space

In releases before z/OS V2R2, the amount of 31-bit virtual storage that was needed by both z/OS UNIX and zFS combined would have exceeded the size of a 2 GB address space. Due to that size limitation, zFS and z/OS UNIX could not coexist in the same address space.

In z/OS V2R2, zFS caches are moved above the 2 GB bar into 64-bit storage. You can now choose to have zFS run in its own colony address space or in the address space that is used by z/OS UNIX, which is OMVS.

When running zFS in the OMVS address space, each file system vnode operation (such as creating a directory entry, removing a directory entry, or reading from a file) will have better overall performance. Each operation will take the same amount of time while inside zFS itself. The performance benefit occurs because z/OS UNIX can call zFS for each operation in a more efficient manner.

Some inherent differences exist when zFS is run in the OMVS address space.
1. MODIFY commands must be passed to zFS through z/OS UNIX. Use the form `MODIFY OMVS,pfs=zfs,cmd`. For more information, see the section on passing a MODIFY command string to a physical file system (PFS) through a logical file system (LFS) in *z/OS MVS System Commands*. This form of the MODIFY command can be used whether zFS is in its own address space or in the OMVS address space.

   Note: When zFS is running in the OMVS address space, any zFS MODIFY commands that are issued through an automated process or system automation must be changed to accommodate the new command format.

2. The CANCEL ZFS command is not available.
3. When the IOEFSPRM configuration file location is defined by the IOEZPRM DD card, it will need to be placed in the OMVS PROC. For more information, see Chapter 12, “The zFS configuration options file (IOEPRMxx or IOEFSPRM),” on page 219.
4. zFS will run under the OMVS user ID instead of the zFS user ID (DFS).
5. You can determine if zFS is in its own address space by issuing D OMVS,PFS. If the output shows an ASNAME value, zFS is running as a colony address space. Otherwise, the lack of an ASNAME value means that zFS is running in the OMVS address space.

Chapter 3. Managing zFS processes

Managing zFS processes includes starting and stopping zFS, as well as determining zFS status.

Starting zFS

zFS is started by z/OS UNIX, based on the FILESYSTYPE statement for zFS in the BPXPRMxx parmlib member. Beginning in z/OS V2R2, if there is no ASNAME keyword on the FILESYSTYPE statement, zFS is started inside the OMVS address space (the address space used by z/OS UNIX). If there is an ASNAME keyword, zFS is started in its own colony address space.

Requirement: Before zFS can start in its own colony address space, a ZFS PROC must be available.

zFS can be started at IPL if the BPXPRMxx parmlib member is in the IEASYSxx parmlib member's OMVS=(*xx*,*yy*) list. To start it later, use the SETOMVS RESET=(*xx*) operator command.

Stopping zFS

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

zFS can be stopped using the MODIFY OMVS,STOPPFS=ZFS operator command. Automatic ownership movement can occur for both the z/OS UNIX owner and the zFS owner. See *z/OS UNIX System Services Planning* for information about the various automove settings for z/OS UNIX file system ownership. When z/OS UNIX notifies zFS that a shutdown is going to occur, zFS aggregate ownership moves to other zFS systems in the shared file system environment. z/OS UNIX then processes its file system ownership changes, or unmounts, as appropriate.

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

```
nn BPXF032D FILESYSTYPE ZFS TERMINATED. REPLY 'R' WHEN READY TO RESTART. REPLY 'I' TO IGNORE.
```

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

To restart zFS, reply `r` to message *nn*. (For example, `r 1,r`). If you want zFS to remain stopped, you can reply `i` to remove the prompt. In this case, zFS can be redefined later using the SETOMVS RESET=(xx)operator command. However, this can result in zFS file systems becoming NOT ACTIVE. An unmount and remount is required to activate a file system that is NOT ACTIVE. If you plan to restart zFS, you should reply `r` to the message.

Note: Stopping zFS can have shared file system (sysplex) implications. See Chapter 5, "Using zFS in a shared file system environment," on page 51 for information about shared file systems.

| If zFS has an internal failure, it typically does not terminate. It might disable an aggregate (see "Diagnosing disabled aggregates" on page 105). If it is a case where
| it does terminate, normally zFS will restart automatically. Otherwise, message BPXF032D (the same message you receive when the MODIFY OMVS,STOPPFS=ZFS operator command is used) is issued and a reply is requested.

On z/OS V1R13 and later systems, if an internal problem occurs, zFS attempts an internal restart. It internally remounts any zFS file systems that were locally mounted, without requiring any support from z/OS UNIX. The zFS ownership for aggregates that are owned on the system that is internally restarted might be moved (by zFS for sysplex-aware file systems) to another system. For more information, refer to Step 10 on page 104.

Determining zFS status

| To determine if zFS is currently active, issue the D OMVS,PFS command. The
| column titled ST (for STatus) will contain an A if zFS is active. It will contain an S
| (Stopped) if it is not.

To display zFS internal restart information, issue the MODIFY ZFS,QUERY,STATUS operator command.

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

A Active; the timestamp is the start time of the PFS.

I Inactive. When the PFS is inactive with no timestamp, the PFS address space has not yet started. When the PFS is inactive with timestamp, the PFS has stop at that time.

S Stopped; it is waiting for a reply of `R` to restart or `I` to terminate the PFS.

U Unavailable.

For more information, see *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

Chapter 4. Creating and managing zFS file systems using compatibility mode aggregates

This section discusses creating compatibility mode aggregates and file systems.

Creating a compatibility mode aggregate

A zFS file system is created in a zFS aggregate (which is a VSAM linear data set). If compatibility mode aggregates are used, the aggregate and the file system are created at the same time. For simplicity, we refer to a file system in a compatibility mode aggregate as a compatibility mode file system. A compatibility mode file system is created using the **ioeagfmt** utility (which is described in "ioeagfmt" on page 126).

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

1. First, use IDCAMS to create a VSAM linear data set.

   Note: Carefully consider defining the aggregate as extended format, extended addressability, and with a secondary allocation size. If you do not use these attributes in the beginning, to add them, you will need to define and format a new zFS aggregate, migrate the data from the original file system into the new one, unmount the original, and then mount the new one. You might want to extend beyond the 4 G aggregate size because version 1.5 aggregates can be much larger than version 1.4 aggregates, or because secondary extents are required to dynamically grow the aggregate, and dynamic grow (aggrgrow) is the default. See "Dynamically growing a compatibility mode aggregate" on page 30 for more information.
2. Then format the VSAM linear data set as a compatibility mode aggregate and create a file system in the aggregate using **ioeagfmt** (see "ioeagfmt" on page 126). When using **ioeagfmt**, the user must meet one of the following authorization requirements:
   - Have ALTER authority to the VSAM linear data set.
   - Be UID 0.
   - Have read authority to the SUPERUSER.FILESYS.PFSCTL resource in the RACF UNIXPRIV class.

Beginning in z/OS V2R1, **ioeagfmt** fails if the zFS PFS is not active on the system. In addition, if the zFS started task does not have the TRUSTED attribute or the OPERATIONS attribute, the DFS user ID must have at least ALTER authority to all VSAM linear data sets that contain zFS aggregates. For details, see *z/OS Security Server RACF Security Administrator's Guide*.

You can also create a compatibility mode aggregate using the ISHELL, or the automount facility, or the **zfsadm define** and **zfsadm format** commands. For more information about ISHELL, see *z/OS V2R2.0 UNIX System Services User's Guide*. For more information about automount, see *z/OS UNIX System Services Planning*. For more information about the **zfsadm define** command, see "zfsadm define" on page 176. For more information about the **zfsadm format** command, see "zfsadm format" on page 188.

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

Rule: The Control Interval (CI) size of a VSAM linear data set that is formatted as a zFS aggregate must be 4 K. This is the default for IDCAMS. As such, it is not specified in Figure 2, which 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
//SYSUDUMP DD     SYSOUT=H
//AMSDUMP  DD     SYSOUT=H
//DASD0    DD     DISP=OLD,UNIT=3390,VOL=SER=PRV000
//SYSIN    DD     *
     DEFINE CLUSTER (NAME(OMVS.PRV.COMPAT.AGGR001) -
            VOLUMES(PRV000) -
            LINEAR CYL(25 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
//SYSUDUMP DD     SYSOUT=H
//CEEDUMP  DD     SYSOUT=H
//*
```

Figure 2. Example job to create a compatibility mode file system

The `-compat` parameter in the CREATE step tells **`ioeagfmt`** to create a compatibility mode file system. As of z/OS V2R1, the `-compat` parameter is optional, and zFS always formats a compatibility mode file system. The result of this job is a VSAM linear data set that is formatted as a zFS aggregate and contains one zFS file system. The zFS file system has the same name as the zFS aggregate (and the VSAM linear data set). The size of the zFS file system (that is, its available free space) is based on the size of the aggregate.

```
//USERIDA JOB ,'Compatibility Mode',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//SYSIN DD *
     DEFINE CLUSTER (NAME(OMVS.PRV.COMPAT.AGGR001) -
            VOLUMES(PRV000) -
            LINEAR CYL(25 10) SHAREOPTIONS(3))
/*
//CREATE EXEC PGM=IOEFSUTL,REGION=0M,
// PARM=('format -aggregate OMVS.PRV.COMPAT.AGGR001')
//SYSPRINT DD SYSOUT=H
//*
```

Figure 3. Example job to create a compatibility mode file system using IOEFSUTL

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

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

- If you specify a number that is less than (or equal to) the number of blocks that fits into the primary allocation, the primary allocation size is used.
- If you specify a number that is larger than the number of 8 KB blocks that fits into the primary allocation, the VSAM linear data set is extended to the size specified if the total size will fit in the primary allocation and a single extension.

A secondary extension cannot be used; instead, see "Growing a compatibility mode aggregate" on page 30. The single extension must be no larger than a single volume. This occurs during its initial formatting. Sufficient space must be available on the volume. Multiple volumes can be specified on the DEFINE of the VSAM linear data set. The multiple volumes are used during extension of the data set at a later time. If you want to create a multi-volume data set initially that is larger than two volumes, see "Creating a multi-volume compatibility mode aggregate" on page 31. DFSMS decides when to allocate on these volumes during extension. Any VSAM linear data set greater than 4 GB can be specified by using the extended format and extended addressability capability in the data class of the data set. See *z/OS DFSMS Using Data Sets* for information about VSAM data sets greater than 4 GB in size.

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

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

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

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

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

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

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

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

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

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

Using version 1.5 aggregates and extended (v5) directories

CAUTION:
Do not use zFS version 1.5 aggregates until you have finished migrating all of your systems to z/OS V2R1 or later. Version 1.5 aggregates are not supported on releases prior to z/OS V2R1. All systems in a sypslex must be a V2R1 level or later before any version 1.5 aggregates on any system in the sysplex are implemented.

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

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

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

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

Creating a version 1.5 aggregate

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

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

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

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

The zFS format utilities **ioeagfmt** and **ioefsutl format** both request the value of the `format_aggrversion` configuration option from the zFS kernel when determining the default aggregate version for the format. If the zFS PFS is down, both utilities will simply fail. Formatting of a version 5 aggregate is not allowed when a z/OS V1R13 system is in a shared file system environment when using the batch utility **ioeagfmt**, the **zfsadm format** command or the Format Aggregate API.

Following is an example of a job to create and format a version 1.5 aggregate:

```
//USERIDA JOB ,'Compatibility Mode',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//AMSDUMP DD SYSOUT=H
//DASD0 DD DISP=OLD,UNIT=3390,VOL=SER=PRV000
//SYSIN DD *
     DEFINE CLUSTER (NAME(OMVS.PRV.COMPAT.AGGR001) -
            VOLUMES(PRV000) -
            LINEAR CYL(25 10) SHAREOPTIONS(3))
/*
//CREATE EXEC PGM=IOEFSUTL,REGION=0M,
// PARM=('format -aggregate OMVS.PRV.COMPAT.AGGR001 -version5')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*
```

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

```
# zfsadm define -aggr OMVS.PRV.ZFS.AGGR005.LDS0005 -volumes PRV000 -cyl 10 5
IOEZ00248I VSAM linear dataset OMVS.PRV.ZFS.AGGR005.LDS0005 successfully created.
# zfsadm format -aggr OMVS.PRV.ZFS.AGGR005.LDS0005 -version5
IOEZ00077I HFS-compatibility aggregate OMVS.PRV.ZFS.AGGR005.LDS0005 has
been successfully created
```

Converting an existing aggregate to version 1.5

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

- Explicitly, for a mounted aggregate using the **zfsadm convert** `-aggrversion` command, or
- Automatically, on mount when the `change_aggrversion_on_mount` configuration option is on (set in IOEPRMxx or using the **zfsadm config** command), or
- Automatically, on mount when the `converttov5` configuration option is on (set in IOEPRMxx or using the **zfsadm config** command), or

- Automatically, on mount when the CONVERTTOV5 MOUNT PARM is specified, or
- Offline, using the IOEFSUTL converttov5 batch utility with the `-aggrversion_only` option.

Note: The CONVERTTOV5 option and MOUNT PARM will also cause accessed directories to be converted to extended (v5) directories after the aggregate is converted to version 1.5.

An aggregate is not automatically changed if the NOCONVERTTOV5 MOUNT PARM is specified. An aggregate is not explicitly or automatically changed if there are earlier release systems (prior to z/OS V2R1) in the shared file system environment.

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

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

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

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

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

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

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

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

Converting a directory from version4 to an extended (v5) directory requires both versions of the directory to exist on disk at the same time, temporarily. If the aggregate becomes full during the allocation of the new directory a dynamic grow will be attempted. See "Dynamically growing a compatibility mode aggregate" on page 30 for information about controlling dynamic growth of an aggregate. If there is not enough space to complete the conversion, the new directory will be deleted and the conversion operation will fail.

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

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

Guidelines for v4 to v5 conversion

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

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

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

After you decide that a file system is going to be converted to v5, you need to decide what conversion method to use. If the file system can be unmounted, the **ioefsutl converttov5** batch utility or MOUNT parameters can be used. If it cannot be unmounted and it is not exported by NFS or SMB servers, use the **zfsadm convert** command. If it is exported by NFS or SMB servers, add the `converttov5` attribute to the mounted aggregate. See "Changing zFS attributes on a mounted zFS compatibility mode file system" on page 40 for instructions about how to add the `converttov5` attribute to the mounted file system.

Migrating data to version 1.5 aggregates

Data can be migrated from HFS file systems into a version 1.5 aggregate in much the same manner as it would be migrated into a version 1.4 aggregate. You can also copy data from a version 1.4 aggregate to a version 1.5 aggregate with the z/OS UNIX shell command **pax**. For more information, see Chapter 7, "Migrating data from HFS or zFS to zFS," on page 65.

Growing a compatibility mode aggregate

If a compatibility mode aggregate becomes full, the administrator can grow the aggregate (that is, cause an additional allocation to occur and format it to be part of the aggregate). This is accomplished with the **zfsadm grow** command. There must be space available on the volume to extend the aggregate's VSAM linear data set. The size that is specified on the **zfsadm grow** command must be larger than the current size of the aggregate.

For example, suppose a two cylinder (primary allocation, 3390) aggregate has a total of 180 8-KB blocks and a (potential) secondary allocation of one cylinder. 180 8-KB blocks is 1440 KB. A **zfsadm aggrinfo** command for this aggregate might show 1440 KB. When you issue the **zfsadm grow** command with a larger size, the file system becomes larger because DFSMS is called to allocate the additional DASD space.

```
zfsadm aggrinfo omvs.prv.aggr003.lds0003

OMVS.PRV.AGGR003.LDS0003 (R/W COMP): 1279 K free out of total 1440
zfsadm grow omvs.orv.aggr003.lds0003 -size 1440

IOEZ00173I Aggregate OMVS.PRV.AGGR003.LDS0003 successfully grown
OMVS.PRV.AGGR003.LDS0003 (R/W COMP): 1279 K free out of total 1440
```

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

```
zfsadm grow omvs.prv.aggr003.lds0003 -size 1441

IOEZ00173I Aggregate OMVS.PRV.AGGR003.LDS0003 successfully grown
OMVS.PRV.AGGR003.LDS0003 (R/W COMP): 1999 K free out of total 2160
```

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

Dynamically growing a compatibility mode aggregate

An aggregate can be dynamically grown if it becomes full. The aggregate (that is, the VSAM linear data set) must have secondary allocation that is specified when it is defined and space must be available on the volume. The number of extensions that are allowed is based on DFSMS VSAM rules (see *z/OS DFSMS Using Data Sets*). The aggregate is extended when an operation cannot complete because the aggregate is full. If the extension is successful, the operation is again transparently driven to the application.

An administrator can restrict aggregates from growing dynamically, either on an individual aggregate basis or globally. To restrict dynamic growing of a specific aggregate, use the NOAGGRGROW parameter on the MOUNT command (see

"MOUNT" on page 150). To globally restrict dynamic growing of all aggregates, specify the `aggrgrow=off` option of the IOEFSPRM configurations option file (see "IOEFSPRM" on page 220).

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

Each time zFS formats a portion of the extension or each time zFS dynamically grows the aggregate and formats a portion of the extension, zFS issues message IOEZ00312I, and then issues one of the following messages:
- IOEZ00309I, when successful
- IOEZ00308E, when unsuccessful

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

Creating a multi-volume compatibility mode aggregate

Before you can create a large zFS aggregate (for example, ten full volumes), you must have the following prerequisites:
- Ten empty volumes
- A DFSMS DATACLASS that provides extended addressability (because the total size is greater than 4 GB)
- A JOB that defines and formats the aggregate.

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

you should get 90 x 3336 = 300,240 8-KB blocks per volume and 10 x 300,240 = 3,002,400 8-KB blocks in the aggregate. Figure 4 on page 32 is an example job that defines the VSAM linear data set in the first step and formats it as a zFS aggregate in the second step. The FORMAT step formats the primary allocation (3336 cylinders) and then extends the data set by the `-grow` amount (300,240 8-KB blocks) ten times (one extend for each full volume) until it reaches the total `-size` amount (3,002,400 8 KB blocks).

The example in Figure 4 on page 32 causes the 10 full volumes to be allocated and formatted by using the `-size` and the `-grow` options on the IOEAGFMT step so that the result is a 10-volume (empty) file system. The `-grow` option is needed in order to allow the specification of a grow increment size that is less than the size of a volume.

```
//USERIDA  JOB ,'Multi-Volume',
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE   EXEC   PGM=IDCAMS
//SYSPRINT DD     SYSOUT=H
//SYSUDUMP DD     SYSOUT=H
//AMSDUMP  DD     SYSOUT=H
//SYSIN    DD     *
     DEFINE CLUSTER (NAME(OMVS.VOL10.COMPAT.AGGR001) -
            VOLUMES(PRV000 PRV001 PRV002 PRV003 PRV004 -
                    PRV005 PRV006 PRV007 PRV008 PRV009) -
            DATACLASS(EXTATTR) -
            LINEAR CYL(3336) SHAREOPTIONS(3))
/*
//FORMAT   EXEC   PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate OMVS.VOL10.COMPAT.AGGR001 -compat -size 3002400 -gX
//            row 300240')
//SYSPRINT DD     SYSOUT=H
//STDOUT   DD     SYSOUT=H
//STDERR   DD     SYSOUT=H
//SYSUDUMP DD     SYSOUT=H
//CEEDUMP  DD     SYSOUT=H
//*
```

Figure 4. Example job to create a multi-volume compatibility mode aggregate

As another example, you could define a VSAM linear data set as before with 10 volumes but with a secondary allocation size of 3336 cylinders, as shown in Figure 5. Then you could format only the first volume by leaving out the `-size` and the `-grow` and let zFS dynamic secondary allocation allocate and format the additional volumes (up to 9 more) as needed. The IOEPRMxx aggrgrow configuration option must be on.

```
 //USERIDA JOB ,'Multi-Volume',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//AMSDUMP DD SYSOUT=H
//SYSIN DD *
     DEFINE CLUSTER (NAME(OMVS.VOL10.COMPAT.AGGR001) -
            VOLUMES(PRV000 PRV001 PRV002 PRV003 PRV004 -
                    PRV005 PRV006 PRV007 PRV008 PRV009) -
            DATACLASS(EXTATTR) -
            LINEAR CYL(3336 3336) SHAREOPTIONS(3))
/*
//FORMAT EXEC PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate OMVS.VOL10.COMPAT.AGGR001 -compat')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*
```

Figure 5. Example job to create a multi-volume compatibility mode aggregate, using a secondary allocation size

Adding volumes to a compatibility mode aggregate

To add a candidate volume to a zFS aggregate, use the IDCAMS utility ALTER command with the ADDVOLUMES parameter. Figure 6 shows an example job that adds two volumes to the (SMS-managed) `OMVS.ZFS.AGGR1` zFS aggregate.

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

Figure 6. Example job to add volumes to a zFS aggregate

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

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

Increasing the size of a compatibility mode aggregate

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

- You can grow the aggregate. For more information, see "Growing a compatibility mode aggregate" on page 30.
- If you cannot grow the aggregate (because, for example, there is no more room on the volume), you can add a volume to the aggregate. For more information, see "Adding volumes to a compatibility mode aggregate."
- If you cannot grow the aggregate and you cannot add a volume (because, for example, you do not have any more volumes available), you can copy the aggregate into a larger VSAM linear data set. There are two ways to copy the data:
  - You can copy each file and directory of the zFS aggregate to a larger data set.
  - You can copy the physical blocks of the zFS aggregate to a larger data set.

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

One method to increase the size of a zFS aggregate is to copy each file and directory of the aggregate to a larger data set. Figure 7 on page 34 shows an example of this approach.

```
//SUIMGVMB JOB ,'EXPAND AGGR WITH PAX',
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//* Make sure you have no line numbers in this JCL
//DEFINE   EXEC   PGM=IDCAMS
//SYSPRINT DD     SYSOUT=H
//SYSIN    DD     *
     DEFINE CLUSTER (NAME(PLEX.NEW.AGGR002.LDS0002) -
            LINEAR CYL(100 5) SHAREOPTIONS(3) -
            VOLUMES(CFC000 CFC001))
/*
//FORMAT   EXEC   PGM=IOEAGFMT,REGION=0M,
//* On the next line, aggregate and compat must be lower case
// PARM=('-aggregate PLEX.NEW.AGGR002.LDS0002 -compat')
//SYSPRINT DD     SYSOUT=H
//******************************************************************
//**                                                              **
//**  note - use a + sign at the end of each line to indicate there**
//**         is another line to be processed.                     **
//**         use a ; at the end of each COMMAND                   **
//**                                                              **
//**         a single command can span multiple lines if each line **
//**         ends in a +.  when you have reached the end of the    **
//**         command, terminate the command with a ;              **
//**                                                              **
//******************************************************************
//PAX1     EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD  SYSOUT=*
//SYSEXEC  DD  DSN=SYS1.SBPXEXEC,DISP=SHR
//SYSTSIN  DD  *
 OSHELL /usr/sbin/mount -t ZFS -f PLEX.OLD.AGGR002.LDS0002          +
  /service2                                                        ; +
   /usr/sbin/mount -t ZFS -f PLEX.NEW.AGGR002.LDS0002 /service3    ; +
   cd /service2                                                    ; +
   pax -rwvCMX -p eW . /service3                                   ;
/*
//* The result of these next two steps should show that
//* More free space is available in the new file system
//AGGRINF1 EXEC PGM=IOEZADM,REGION=0M,
// PARM=('aggrinfo PLEX.OLD.AGGR002.LDS0002 -long')
//SYSPRINT DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
/*
//AGGRINF2 EXEC PGM=IOEZADM,REGION=0M,
// PARM=('aggrinfo PLEX.NEW.AGGR002.LDS0002 -long')
//SYSPRINT DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
/*
```

Figure 7. Sample job to copy each file and directory of an aggregate to a larger data set

This approach uses the **pax** command to copy the individual files and directories into an already formatted and empty zFS file system. Both file systems must be mounted. **pax** uses the z/OS UNIX file and directory APIs to read and write each individual file and directory of the hierarchy of the file system. (It does not copy lower mounted file systems because of the `-X` and `-M` options.) You can use the

ISHELL command or the **automount** command with the allocany or allocuser keyword to create the new larger aggregate to copy into with **pax**, because they format the aggregate.

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

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

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

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

Another method to increase the size of a zFS aggregate is to copy the physical blocks of the aggregate to a larger data set using the DFSMS REPRO command. This approach is normally faster than using the **pax** command. However, do not format the target zFS data set before using the REPRO command. Figure 8 on page 36 shows an example of this approach.

```
//SUIMGVMB JOB ,'EXPAND AGGR WITH REPRO',
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE   EXEC   PGM=IDCAMS
//SYSPRINT DD     SYSOUT=H
//SYSIN    DD     *
     DEFINE CLUSTER (NAME(PLEX.NEW.AGGR002.LDS0002) -
            LINEAR CYL(100 5) SHAREOPTIONS(3) -
            VOLUMES(CFC000 CFC001))
/*
//LCAT1    EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//* This step should show a HI-U-RBA of 0
//* for PLEX.NEW.AGGR002.LDS002
//SYSIN    DD  *
          LISTCAT ENTRIES(PLEX.OLD.AGGR002.LDS0002) -
                  ALL
          LISTCAT ENTRIES(PLEX.NEW.AGGR002.LDS0002) -
                  ALL
/*
//REPRO1   EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//* The next line guarantees that the file system is not mounted
//IN1   DD DSN=PLEX.OLD.AGGR002.LDS0002,DISP=OLD
//SYSIN DD *
  REPRO -
     INFILE(IN1) -
     OUTDATASET(PLEX.NEW.AGGR002.LDS0002)
/*
//LCAT2    EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//* This step should show the HI-U-RBA of
//* PLEX.NEW.AGGR002.LDS002 equal to the HI-U-RBA
//* of PLEX.OLD.AGGR002.LDS002
//SYSIN    DD  *
          LISTCAT ENTRIES(PLEX.OLD.AGGR002.LDS0002) -
                  ALL
          LISTCAT ENTRIES(PLEX.NEW.AGGR002.LDS0002) -
                  ALL
/*
```

Figure 8. Sample job to copy the physical blocks of an aggregate to a larger data set

Figure 9 on page 37 shows a zFS file system (PLEX.OLD.AGGR002.LDS0002) that is full and a newly-defined zFS data set (PLEX.NEW.AGGR002.LDS0002 before the REPRO) that is larger. PLEX.NEW.AGGR002.LDS0002 has a larger HI-A-RBA than PLEX.OLD.AGGR002.LDS0002. When the blocks from PLEX.OLD.AGGR002.LDS0002 are copied into PLEX.NEW.AGGR002.LDS0002 using REPRO, the result is PLEX.NEW.AGGR002.LDS0002 after REPRO. There is now room to add data to PLEX.NEW.AGGR002.LDS0002.

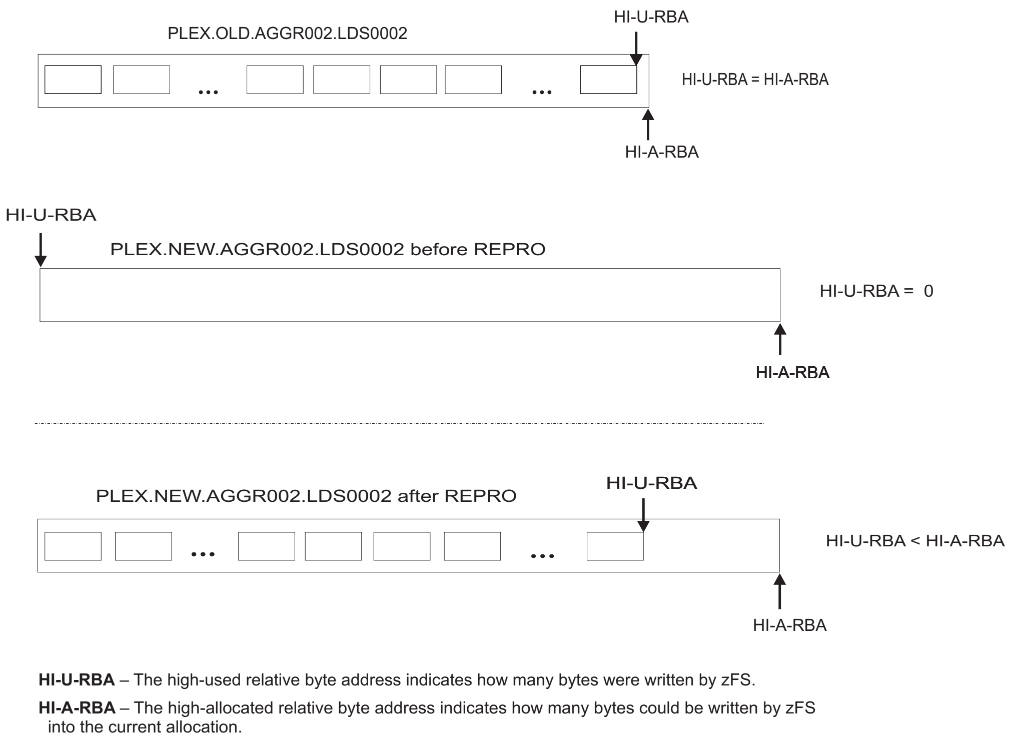
PLEX.OLD.AGGR002.LDS0002
HI-U-RBA
HI-U-RBA = HI-A-RBA
HI-A-RBA
HI-U-RBA
PLEX.NEW.AGGR002.LDS0002 before REPRO
HI-U-RBA = 0
HI-A-RBA
PLEX.NEW.AGGR002.LDS0002 after REPRO
HI-U-RBA
HI-U-RBA < HI-A-RBA
HI-A-RBA
HI-U-RBA – The high-used relative byte address indicates how many bytes were written by zFS.
HI-A-RBA – The high-allocated relative byte address indicates how many bytes could be written by zFS into the current allocation.

Figure 9. Copying blocks from a full zFS data set into a larger data set

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

Follow these guidelines:

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

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

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

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

Decreasing the size of a compatibility mode aggregate

You can decrease the size of a zFS aggregate using the **pax** command. Figure 10 shows a sample job.

```
//SUIMGVMB JOB ,'SHRINK AGGR WITH PAX',
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//* Make sure you have no line numbers in this JCL
//DEFINE   EXEC   PGM=IDCAMS
//SYSPRINT DD     SYSOUT=H
//SYSIN    DD     *
     DEFINE CLUSTER (NAME(PLEX.NEW.AGGR002.LDS0002) -
            LINEAR CYL(25 5) SHAREOPTIONS(3) -
            VOLUMES(CFC000 CFC001))
/*
//FORMAT   EXEC   PGM=IOEAGFMT,REGION=0M,
//* On the next line, aggregate and compat must be lower case
// PARM=('-aggregate PLEX.NEW.AGGR002.LDS0002 -compat')
//SYSPRINT DD     SYSOUT=H
//*****************************************************************
//**                                                             **
//**  note - use a + sign at the end of each line to indicate there**
//**         is another line to be processed.                    **
//**         use a ; at the end of each COMMAND                  **
//**                                                             **
//**         a single command can span multiple lines if each line **
//**         ends in a +.  when you have reached the end of the    **
//**         command, terminate the command with a ;             **
//**                                                             **
//*****************************************************************
//PAX1     EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD  SYSOUT=*
//SYSEXEC  DD  DSN=SYS1.SBPXEXEC,DISP=SHR
//SYSTSIN  DD  *
 OSHELL /usr/sbin/mount -t ZFS -f PLEX.OLD.AGGR002.LDS0002          +
  /service2                                                       ; +
   /usr/sbin/mount -t ZFS -f PLEX.NEW.AGGR002.LDS0002 /service3    ; +
   cd /service2                                                   ; +
   pax -rwvCMX -p eW . /service3                                  ;
//* The result of these next two steps should show that
//* the new file system is smaller
//AGGRINF1 EXEC PGM=IOEZADM,REGION=0M,
// PARM=('aggrinfo PLEX.OLD.AGGR002.LDS0002 -long')
//SYSPRINT DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
/*
//AGGRINF2 EXEC PGM=IOEZADM,REGION=0M,
// PARM=('aggrinfo PLEX.NEW.AGGR002.LDS0002 -long')
//SYSPRINT DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
/*
```

Figure 10. Sample job to decrease the size of an aggregate

This approach uses the **pax** command to copy the individual files and directories into an already formatted and empty zFS file system. Both file systems must be mounted. **pax** uses the z/OS UNIX file and directory APIs to read and write each individual file and directory of the hierarchy of the file system. (It does not copy lower mounted file systems because of the `-X` and `-M` options.)

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

Renaming or deleting a compatibility mode aggregate

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

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

The example in Figure 11 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.*)
/*
```

Figure 11. Example job to reconcile the file system and aggregate name

To delete a compatibility mode aggregate, use the IDCAMS utility DELETE command. You cannot delete an aggregate if it is mounted. Figure 12 shows a sample job that deletes both the cluster name and the data component.

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

Figure 12. Example job to delete a compatibility mode aggregate

See *z/OS DFSMS Access Method Services Commands* for information and restrictions on IDCAMS ALTER NEWNAME and DELETE.

Changing zFS attributes on a mounted zFS compatibility mode file system

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

- AGGRFULL
- AGGRGROW
- CONVERTTOV5
- FSFULL

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

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

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

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

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

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

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

You can query the current default value of a zFS attribute by issuing the **zfsadm configquery** command. For example, to query the default value of the following attributes, you can issue the following commands:

```
zfsadm configquery -aggrfull
zfsadm configquery -converttov5
zfsadm configquery -fsfull
zfsadm configquery -aggrgrow
zfsadm configquery -sysplex_filesys_sharemode
```

You can change a zFS attribute on a mounted file system. To do so, take an appropriate action, as previously described for the attribute that you want to change. For example, to change the NORWSHARE attribute of a compatibility mode file system to RWSHARE, you can move the z/OS UNIX ownership of that file system to a different system that specifies RWSHARE as the zFS default.

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

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

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

Unmounting zFS file systems before copying or moving

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

When a zFS aggregate is unmounted (detached), the system name and the time stamp are cleared. In this case, the next mount does not wait because zFS knows that the aggregate is not currently mounted. If the aggregate is being mounted on a different member in the same sysplex after a failure, zFS does not wait because it recognizes that this is a different system that is in the same sysplex.

As a result, you can cause zFS to wait during mount unnecessarily and you can experience z/OS UNIX latch contention if you fail to unmount (detach) a zFS aggregate before copying it or moving it to another system.

Understanding zFS disk space allocation

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

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

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

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

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

This failure can occur because you are trying to remove an entry from a directory that was created before z/OS V1R13 and is smaller than 7 KB, so it is stored in fragments. But the file system is zFS-owned on a z/OS V1R13 or later system and needs a free 8-KB block to do the remove. To resolve this problem, you must explicitly grow the file system in order to make free 8-KB blocks available. You can do this even if the zFS file system data set does not have a secondary allocation size specified. Free space on the volume is required. For example:

```
# rm /service6/testdir2/filea
rm: FSUM9195 cannot unlink entry "/service6/testdir2/filea":
EDC5133I No space left on device.
```

```
# zfsadm aggrinfo PLEX.JMS.AGGR006.LDS0006
PLEX.JMS.AGGR006.LDS0006 (R/W COMP): 21 K free out of total 7200
# zfsadm grow PLEX.JMS.AGGR006.LDS0006 7920
IOEZ00173I Aggregate PLEX.JMS.AGGR006.LDS0006 successfully grown
PLEX.JMS.AGGR006.LDS0006 (R/W COMP):741 K free out of total 7920
# rm /service6/testdir2/filea
#
```

If you need to add a volume, you can add one using the IDCAMS ALTER command with the ADDVOLUMES option. For more information, see “Adding volumes to a compatibility mode aggregate” on page 33.

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

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

Bitmap
: Lists the blocks that are free on disk. The file size depends on the size of the aggregate.

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

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

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

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

zFS stores files on disk in one of three ways:

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

fragmented
: On systems before z/OS V1R13, if the file is 7 KB or less and has never been larger than 7 KB, zFS stores it in 1-KB fragments; as such, it is stored in part of an 8-KB block. Multiple small files can share the same 8-KB

block on disk. On z/OS releases z/OS V1R13 and laterYeah systems, zFS no longer stores files in 1-KB fragments.

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

How data is stored on systems before z/OS V1R13

On systems before z/OS V1R13, zFS can store data in fragmented blocks to conserve disk space. On these systems, each small file does not need to use a full 8-KB block of disk space. However, as a result of this method of storing data, a problem can occur when data is stored using zFS. That is, the amount of free space that is displayed by the z/OS UNIX **`df`** command might not give the entire picture of free space. The **`df -k`** command displays free space in a file system in 1-KB units. In zFS, this space is a combination of full 8-KB blocks plus the free 1-KB fragments in fragmented blocks. For example, as Figure 13 shows, if there were two 8-KB blocks and twenty 1-KB blocks that are left, **`df -k`** reports 36 KB available.

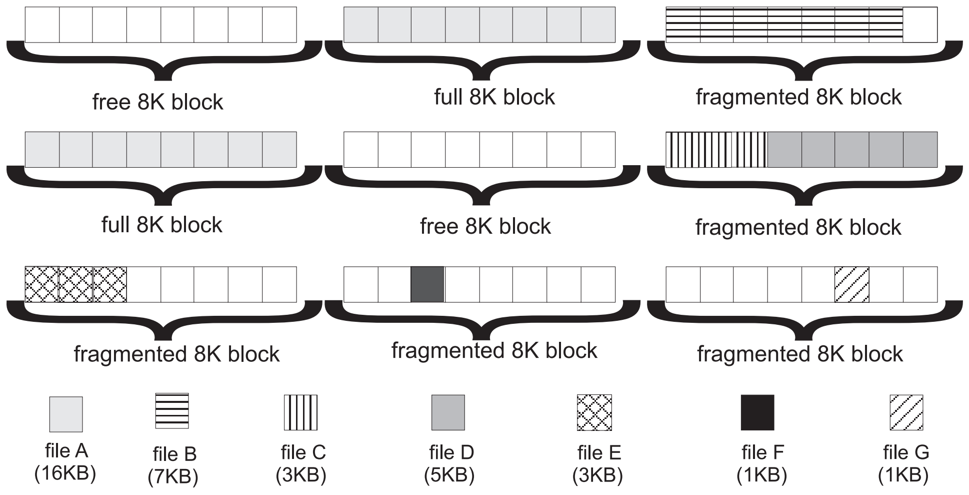

Figure 13. Disk space allocation example 1

Because this is a combination of 8-KB blocks and 1-KB blocks, it is possible that many 1-KB blocks are available but no 8-KB blocks remain. As shown in Figure 14 on page 45 for example, if there were 0 8-KB blocks left and 20 1-KB blocks available, **`df -k`** reports 20 KB available. If you try to create a 10-KB file, you might think that there is plenty of space. However, a 10-KB file is larger than 7 KB, and therefore uses full 8 KB blocks. Because there are no 8-KB blocks available, there is no room for a 10 KB file, even though there is 20-KB free space.

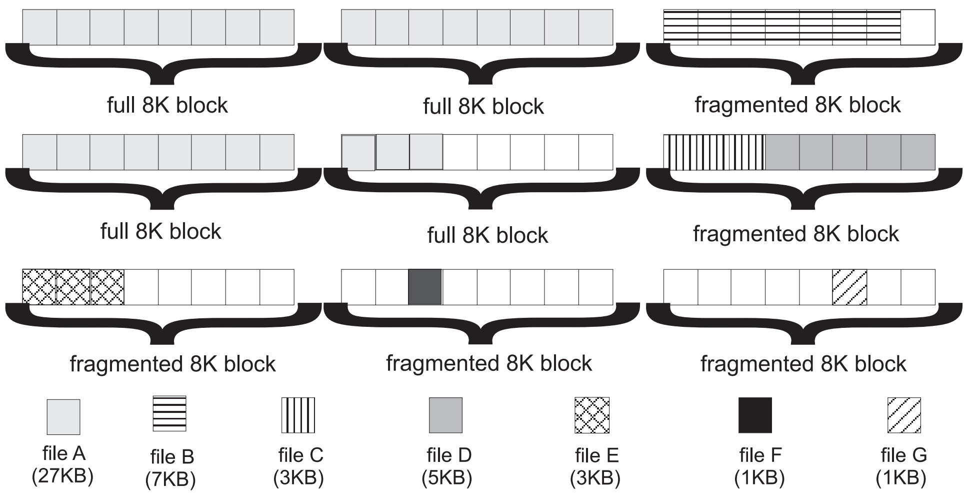

Figure 14. Disk space allocation example 2

Other rules can further restrict how free space is used. A file that is 7 KB must be stored in 7 contiguous fragments. Therefore, even if there is 20 KB available in the file system, if there is no fragmented block with 7 contiguous 1-KB blocks available, the file system will report that there is no space for the file. Also, a file that is stored as fragments cannot share the same 8-KB block as a directory stored as fragments.

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

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

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

| ## Support for type 30 SMF record

| The type 30 SMF record provides accounting information. z/OS UNIX contributes
| to them, in part, by providing a count of the number of blocks that are read from
| file system disk blocks, or written to file system disk blocks, during each operation
| performed in a UNIX file system by a user or an application. The SMF30OFR and
| SMF30OFW fields of the SMF record contain these counts. The zFS PFS provides
| the count of blocks that are involved in these I/O operations to z/OS UNIX in the
| OSI control block fields `readibc` and `writeibc`.

| Due to the aggressive caching that zFS does with the contents of the disk blocks, it
| is not possible for zFS to provide an exact count of actual I/O operations that are

done by each user or application. Instead, zFS provides a weighted cost estimation of the number of disk blocks an operation could read or write. This method of counting the blocks is not the same as that used by HFS, so comparisons of HFS versus zFS file systems will not be accurate. This method of counting the blocks should be consistent enough to allow the comparison of two users or applications accessing the same zFS file system. This will be true even if the file system is mounted RWSHARE and accessed from two different systems that are sharing it.

Sharing zFS data in a non-shared file system sysplex

For information about sharing zFS data in a shared file system in a multisystem sysplex environment, see Chapter 5, "Using zFS in a shared file system environment," on page 51 and review "Unmounting zFS file systems before copying or moving" on page 41.

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

Minimum and maximum file system sizes

The minimum zFS compatibility mode aggregate size is six 3390 tracks, which hold thirty-six 8 KB blocks (six 8 KB blocks per track × 6 tracks). In the example in Figure 15, DFSMS allocates 7 tracks. Six 8-KB blocks per track x 7 tracks is 42 8-KB blocks or 336 KB. This only leaves 184 KB of free space available for files and directories. Small file systems tend to fill up quickly because of block and fragment allocation and can appear to have free space when they really do not. (For more information, see "Understanding zFS disk space allocation" on page 42). Using such small file systems is not a good idea. You can permit the file system to grow automatically (you must have `aggrgrow=on` in the IOEFSPRM file, which is the default, or in the MOUNT PARM. You must also have a secondary allocation specified on the **zfsadm define** command, which is specified as 5 in Figure 15). 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.AGGR006.LDS0006 -volumes CFC000 -tracks 6 6

IOEZ00248I VSAM linear dataset PLEX.JMS.AGGR006.LDS0006 successfully created.

# zfsadm format PLEX.JMS.AGGR006.LDS0006

IOEZ00077I HFS-compatibility aggregate PLEX.JMS.AGGR006.LDS0006 has been successfully created
# /usr/sbin/mount -t ZFS -f PLEX.JMS.AGGR006.LDS0006 -o 'RWSHARE' /service6
# zfsadm aggrinfo PLEX.JMS.AGGR006.LDS0006 -long
PLEX.JMS.AGGR006.LDS0006 (R/W COMP): 184 K free out of total 336
version 1.4
auditfid C3C6C3F0 F0F200CC 0000
sysplex-aware
           23 free 8k blocks;          0 free 1K fragments
          112 K log file;              8 K filesystem table
            8 K bitmap file
```

Figure 15. Example of a secondary zfsadm define command

Version 1.5 aggregates

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

```
262668 cylinders per volume
x 90 blocks per cylinder
x 8KB per block
x 59 volumes
--------------------------
10641 GB or 10.39 TB
```

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

Version 1.4 aggregates

For a version 1.4 aggregate, the architected maximum size for compatibility mode aggregates is approximately 4 TB (1 KB x 4 GB). If you use 3390 DASD that has 65,520 cylinders per volume, you can create a compatibility mode aggregate of about 2,850,088,550,400 bytes.

```
65520 cylinders per volume
x 90 blocks per cylinder
x 8KB per block
x 59 volumes
--------------------------
2654 GB or 2.59 TB
```

Restriction: A zFS version 1.4 compatibility mode aggregate is limited to 4 TB even on extended address volume (EAV) devices. A zFS version 1.5 compatibility mode aggregate is limited to 16 TB even on extended address volume (EAV) devices.

The maximum number of objects (files, directories, and ACLs) in a zFS file system is 4 G. The maximum size of a file is approximately 4 TB. The maximum size of a directory is 4 GB. There is a limit of 65,533 (64K -1) subdirectories in a directory for a v4 directory. There is a limit of 4,294,967,293 (4G-1) subdirectories in a directory for an extended (v5) directory. The maximum number of names in a directory is dependent on the length of the names. However, there is a known performance problem when you have a large number of names (hundreds of thousands or millions) in a single zFS v4 directory. For best performance, use an extended (v5) directory in a version 1.5 aggregate. See “Using version 1.5 aggregates and extended (v5) directories” on page 26 for information about extended (v5) directories. If you must use a version 1.4 aggregate because you are still running releases prior to z/OS V2R1, try to spread names among many directories.

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

v4 directory considerations

For v4 directions only, if you have long response times, you can get a first indication whether you might have a directory size problem by examining the output of the MODIFY ZFS,QUERY,KN operator command or the z/OS UNIX **`zfsadm query -knpfs`** command. Look at the **Avg Time** field on the lines for operations that require zFS to search through names of a directory (for example, zfs_lookup, zfs_create, or zfs_remove). Typically, the average times should be on the order of a few milliseconds. If they are relatively large (perhaps ten to a hundred times larger than that), it is possible that you have a directory that is too large and is causing performance problems. You can use the **largedir.pl** command to help determine which directories are large. It reports any zFS directories that have a large size. The **largedir.pl** command is available on the z/OS UNIX System Services Tools and Toys web page (http://www.ibm.com/systems/z/os/zos/features/unix/bpxa1ty2.html).

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

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

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

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

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

 - or -


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

 - or -

zfsadm convert -path /zfsmnt5/testdir
```

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

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

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

See Chapter 7, “Migrating data from HFS or zFS to zFS,” on page 65 for information about copying one file system to another. For information about converting an existing file system to version 1.5, see “Using version 1.5 aggregates and extended (v5) directories” on page 26.

When you must have many file names in a single directory, it is best to use a version 1.5 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 specification of SYSPLEX(YES) in the BPXPRMxx parmlib member. 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.

To better understand the terminology and concepts in this section, review "Terminology and concepts" on page 4.

Overview of the shared file system environment

In a shared file system environment, file systems that are mounted read-only are always sysplex-aware.

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

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

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

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

Read-only mounted file systems

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

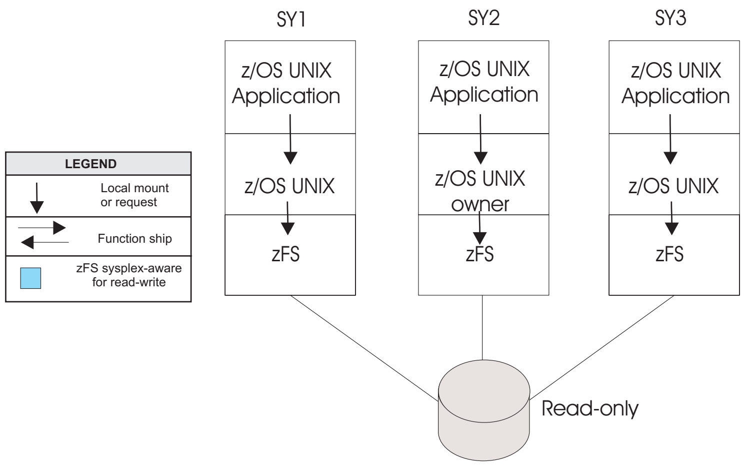

Figure 16. Sysplex-aware file system (read-only)

For read-only mounted file systems, file requests are sent directly to the local physical file system, which directly reads the file system data on DASD (see Figure 16). 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 file systems with different levels of sysplex-awareness

zFS allows individual zFS read/write file systems to be mounted sysplex-aware or non-sysplex aware. During mount processing, the sysplex-awareness of an individual zFS read/write file system can be controlled by the value that is specified on the mount PARM for that file system or by the `sysplex_filesys_sharemode` option that is specified in IOEFSPRM. Table 1 summarizes how the sysplex awareness is determined.

Table 1. Determining sysplex-awareness for zFS read/write file systems

MOUNT PARM	Resulting awareness of the zFS read/write file system
RWSHARE	Sysplex-aware
NORWSHARE	Non-sysplex aware
None specified	Determined by the value, if any, specified on the `sysplex_filesys_sharemode` option. **rwshare** File system is sysplex-aware **norwshare** File system is non-sysplex aware **not specified** File system defaults to be non-sysplex aware

| Figure 17 on page 53 shows one file system that is mounted NORWSHARE and
| the other mounted RWSHARE. They are both owned by z/OS UNIX on SY2. The NORWSHARE file system is a non-sysplex aware file system; it is only locally mounted on the z/OS UNIX owner and requests from z/OS UNIX clients are function shipped to the z/OS UNIX owner by z/OS UNIX.

- A **df –v** command for the NORWSHARE file system (FS1) from SY1 would display `Client=Y`, or a D OMVS,F command would display `CLIENT=YES`. The other file system is mounted RWSHARE. It is a sysplex-aware file system; it is locally mounted on all systems and z/OS UNIX does not normally function ship requests to the z/OS UNIX owner.
- A **df –v** command for the RWSHARE file system (FS2) from SY1 would display `Client=N`, or a D OMVS,F command would display `CLIENT=N`.

The following example shows the mount of a zFS read/write- file system with a mount PARM of RWSHARE:

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

zFS-enhanced sysplex-aware support

Beginning in z/OS V1R13, zFS provides enhanced sysplex-aware support. When a zFS read/write file system is mounted sysplex-aware in a shared file system environment where all systems are running z/OS V1R13 or later, zFS can directly read and write zFS data from all of the V1R13 or later systems. If both the owning system and the requesting system are running z/OS V1R13 or later (and the file system is sysplex-aware), zFS directly accesses the file system. While zFS data is directly read and written, zFS metadata is normally read and written through the zFS owning system (SY2 in Figure 17). In some cases, zFS metadata can be directly read.

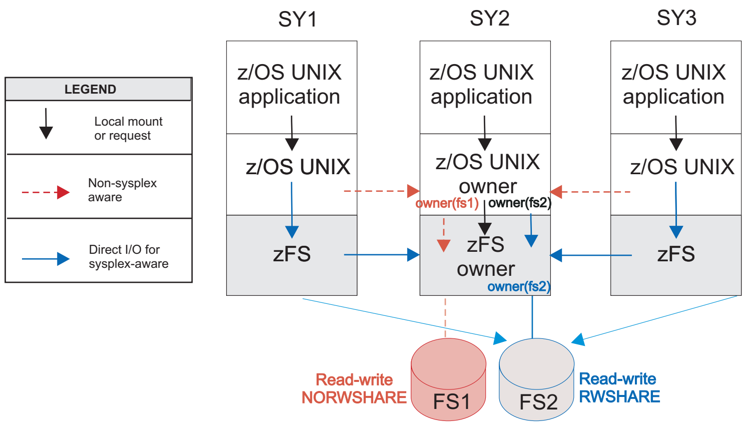

Figure 17. zFS read/write file systems sysplex-aware and non-sysplex aware on a file system basis. FS2 is being directly accessed from all z/OS V1R13 or later systems.

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

zFS ownership versus z/OS UNIX ownership of file systems

For zFS read/write sysplex-aware file systems, zFS takes responsibility for determining how to access the data. This means that zFS must have the concept of a file system owner to coordinate file requests. That system is the *zFS owner*. z/OS UNIX has its indication of owner, which is called the *z/OS UNIX owner*. The zFS owner is independent of the z/OS UNIX owner. The zFS owner is the system that coordinates file access. The z/OS UNIX owner generally does not have any performance implications when zFS runs sysplex-aware because file requests are sent to the local zFS rather than being function shipped to the z/OS UNIX owner. There are some cases when the z/OS UNIX owner is relevant (see "When is the z/OS UNIX owner important?" on page 56).

In Figure 18, 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 18 shows the situation after SY2 has come back up. (zFS on SY1 communicates directly with zFS on SY3.) The fact that SY1 is the z/OS UNIX owner is not important for performance in this case.

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

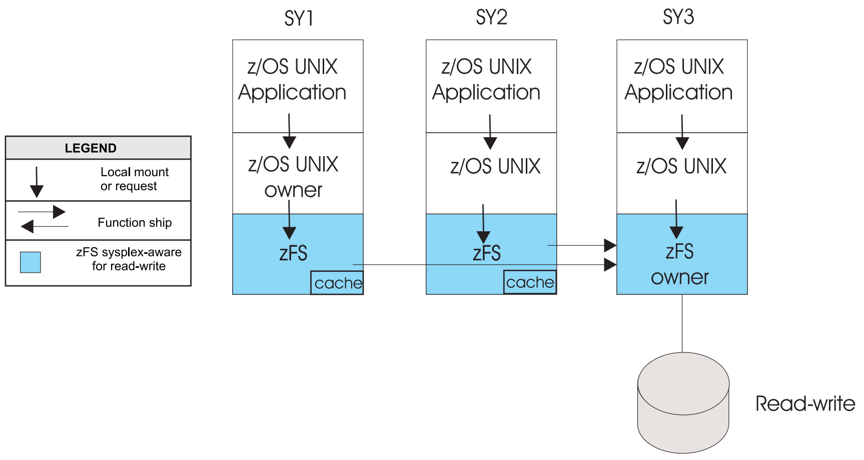

Figure 18. zFS sysplex-aware with new owner

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** shell command
- D OMVS,F operator command
- F BPXOINIT,FILESYS=D,ALL operator command

Figure 19 on page 55 shows the output of the **zfsadm lsaggr** command and the **df -v** command after the file system was mounted.

```
# 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 19. **zfsadm lsaggr** *and* **df -v** *output after mount*

Figure 20 shows the output of the D OMVS,F command after the file system was mounted.

```
D OMVS,F
BPXO045I 14.38.11 DISPLAY OMVS
OMVS     000E ACTIVE             OMVS=(P0,VM)
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
ZFS           26 ACTIVE                      RDWR  02/02/2011  L=55
  NAME=PLEX.JMS.AGGR008.LARGE08                    14.37.44    Q=0
  PATH=/zfsmnt5
  OWNER=SY2      AUTOMOVE=Y CLIENT=N
```

Figure 20. D OMVS,F output after mount

Figure 21 shows the output of the **zfsadm lsaggr** command and the **df -v** command after the file system was moved (as shown in Figure 18 on page 54) by both z/OS UNIX and zFS and SY2 has come back up. The **zfsadm lsaggr** and **df -v** commands are issued from SY2:

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

Figure 22 on page 56 shows the output of the D OMVS,F operator command after the file system was moved. Notice two important points:

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

```
D OMVS,F
BPXO045I 14.38.11 DISPLAY OMVS
OMVS     000E ACTIVE            OMVS=(P0,VM)
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
ZFS           26 ACTIVE                       RDWR  02/02/2011  L=55
  NAME=PLEX.JMS.AGGR008.LARGE08                     14.37.44    Q=0
  PATH=/zfsmnt5
  OWNER=SY1      AUTOMOVE=Y CLIENT=N
```

Figure 22. D OMVS,F output after movement

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

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

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

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

When is the z/OS UNIX owner important?

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

When a zFS sysplex-aware file system is mounted, z/OS UNIX causes the file system to be locally mounted on each system (where zFS is running sysplex-aware). These are called *catch-up mounts*. If a local catch-up mount fails (for example, because the DASD is not accessible from that system), then z/OS UNIX treats that system (such as SY1) as a client and function ships requests to the z/OS UNIX owner (SY2). The system (SY1) might issue message BPXF221I. In this case, a **df -v** command issued from SY1 indicates Client=Y for that file system. In turn, zFS directly accesses the file system and function ships metadata updates to the zFS owner, if the zFS owner is a different system than the z/OS UNIX owner—in this case, it is not different (for example, see Figure 23 on page 57).

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

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

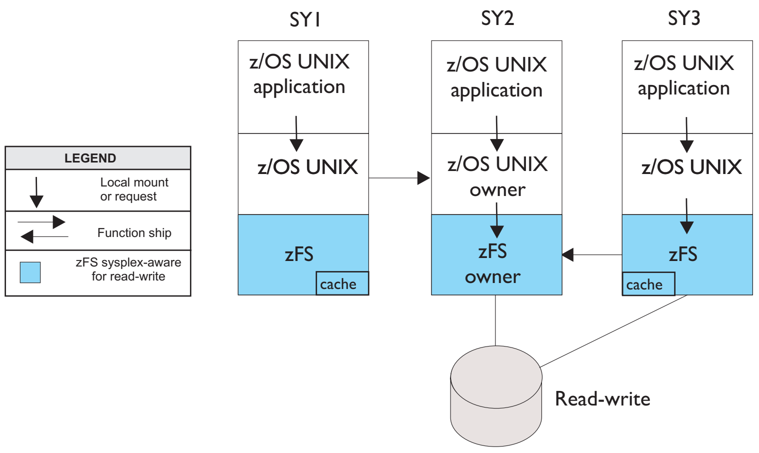

Figure 23. File system ownership when mount fails

Dynamic movement of the zFS owner

For zFS read/write sysplex-aware file systems, an important aspect of performance is which system is the zFS owner. The zFS owner is the system that does metadata updates to the file system. zFS automatically moves the zFS owner among zFS systems, based on the amount of activity at the zFS owner from each system. The frequency of the dynamic ownership movement can vary, depending on the level of zFS. On z/OS V1R13 and later systems, ownership moves less often than on systems that are running previous levels of z/OS.

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

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

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

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

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

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

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

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

Tip: Mount system-specific zFS file systems with UNMOUNT instead of NOAUTOMOVE.

One way to think of the relationship between z/OS UNIX ownership movement and zFS aggregate ownership movement is:
- z/OS UNIX controls whether there is any access at all
- zFS ownership controls which system updates the metadata.

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

Considerations when using zFS in a shared file system environment

The following considerations apply when using zFS in a sysplex in shared file system mode:
- The file system hierarchy appears different when viewed from systems with zFS mounted file systems than it does from those systems not running zFS. The path name traversal through zFS mount points have different results in such cases because the zFS file system is not mounted on those systems not running zFS.
- zFS file systems that are owned by another system are accessible from a member of the sysplex that is running zFS.
- zFS compatibility mode file systems can be automoved and automounted. A zFS compatibility mode file system can only be automoved to a system where zFS is running.
- To share IOEFSPRM across a sysplex, configuration options that specify data set names should use system symbols in the names. This needs to be done for data sets that zFS writes into, such as the data sets specified by configuration options `trace_dsn` or `msg_output_dsn`. It is also allowed, but not necessary, to use system symbols in the names of data sets that zFS reads data from, such as the data set specified by the configuration option `debug_settings_dsn`. For more information, see Chapter 12, “The zFS configuration options file (IOEPRMxx or IOEFSPRM),” on page 219.

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

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

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

As a preferred alternative to the IOEZPRM DDNAME specification, the IOEFSPRM member can be specified as a true PARMLIB member. In this case, the member has the name IOEPRMxx, where xx is specified in the parmlib member list. It is possible to have multiple IOEPRMxx members and it is also possible to have an IOEPRMxx member that are shared among all members of the sysplex and another IOEPRMxx member that contains options that are specific to a particular sysplex member. See "IOEFSPRM" on page 220 for more information about IOEPRMxx.

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

- All systems running zFS see zFS compatibility mode file systems. The file system hierarchy appears differently when viewed from systems with zFS mounted compatibility mode file systems than it does from those systems that are not running zFS. The path name traversal through zFS mount points have different results in such cases because the zFS compatibility mode file system is not mounted on those systems that are not running zFS.
- If a system running zFS is brought down:
  - zFS compatibility mode file systems owned by the system that can be automoved are automoved to another system running zFS. If this function fails to find another owner, the file system becomes unowned. IBM recommends mounting zFS file systems with UNMOUNT instead of NOAUTOMOVE.
  - zFS compatibility mode file systems that are NOAUTOMOVE, become unowned.
  - zFS compatibility mode file systems that are unowned are not visible in the file system hierarchy, but can be seen from a D OMVS,F command. To recover a zFS compatibility mode file system that is mounted and unowned, the zFS compatibility mode file system must be unmounted.
  - The unowned zFS compatibility mode file systems can be recovered if the original owning system is brought back into the sysplex.
- If zFS is brought down on one system in the sysplex:
  - zFS compatibility mode file systems owned by the system that can be automoved are automoved to another system running zFS. If this function does not find another z/OS UNIX owner, the zFS compatibility mode file system, and all file systems mounted under it, are unmounted in the sysplex.
  - zFS compatibility mode file systems that are NOAUTOMOVE and, all file systems mounted under them, are unmounted in the sysplex.
  - When zFS is down on one system (SY1) in the sysplex, z/OS UNIX does not function ship any zFS compatibility mode file system that is subsequently mounted on another system. That file system is not visible from SY1. zFS can be brought up again on that system by responding R to the BPXF032D prompt. When this occurs, mounted file system visibility is established by one of the following methods:
    - If the zFS file system is non-sysplex aware, z/OS UNIX function shipping is established
    - If zFS file system is sysplex-aware, the zFS file system is locally mounted

- When a zFS is brought down after a compatibility mode file system is mounted, the file system either continues to be function shipped or becomes function shipped. When zFS is brought back up on that system, the file system either:
  - Continues to be function shipped, when the zFS file system is non-sysplex aware
  - Is locally mounted, when the zFS file system is sysplex-aware

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

Chapter 6. Copying or performing a backup of a zFS

CAUTION:
Do not perform any type of COPY or DUMP operation of DASD that contains a mounted zFS file system that is not quiesced, or that is mounted on a system that is not a member of the same GRS configuration as the system from which the COPY or DUMP operation is being done. Doing so might result in the copy being a corrupted (or unusable) zFS file system. For additional information about DFSMSdss logical DUMP and COPY utilities, see the section on zFS data sets in *z/OS DFSMSdss Storage Administration***.**

You can back up a zFS aggregate using a DFSMSdss logical dump. DFSMSdss automatically performs a quiesce of the mounted zFS aggregate before dumping the data set and an unquiesce when the dump ends. Before performing a backup, review the information in "Unmounting zFS file systems before copying or moving" on page 41 and the following guidelines.

Review the following guidelines before performing a backup of zFS:

1. Do not specify TOL(ENQF) when backing up zFS aggregates because it can cause corruption of the file system.
2. Full volume dumps of volumes that contain mounted zFS file systems will not quiesce the file systems. As a result, all file systems that reside on the volume must be unmounted before performing a full volume dump. For information about logical and full volume dumps, see *z/OS DFSMSdfp Storage Administration*.
3. The term *sysplex* as it applies to zFS means a sysplex that supports the z/OS UNIX shared file system environment. That is, a sysplex that has a BPXPRMxx specification of SYSPLEX(YES).
4. If a quiesce is not done before the backup of a mounted file system, corruption of the file system can result. If you are using a different program or different commands than shown in "Backing up a zFS aggregate" on page 62, 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 24 on page 62.

```
//*-----------------------------------------------------------------
//* 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 24. Steps for quiesce and unquiesce

Backing up a zFS aggregate

Figure 25 shows an example of a job for backing up a zFS aggregate (and all the file systems). Ensure that the size of the target sequential data set has sufficient space. For additional information about the DUMP command and its keywords, see *z/OS DFSMSdfp Storage Administration*.

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

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

Leading blanks are required before the control statements (DUMP, RESET, OUTDD).

Figure 25. Job to back up a zFS aggregate

Restoring an aggregate with DFSMSdss logical restore

Use DFSMSdss logical restore to restore a zFS aggregate. If the original aggregate (in the example, `hlq.ZFS.AGGR004`) still exists, the aggregate is restored into a new aggregate (in the example, OMVS.PRV.AGGR005.LDS0005). Figure 26 is an example of a job to restore a zFS aggregate.

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

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

Figure 26. Job to restore a zFS aggregate

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

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

Figure 27 is an example of a job to perform a logical restore of a zFS aggregate using DFSMSdss by replacing the existing aggregate. The backup is restored into the original aggregate (in this case, hlq.ZFS.AGGR004). The aggregate cannot be mounted (or attached) during the restore operation.

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

Figure 27. Job to restore a zFS aggregate with replace

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

For more information about DFSMSdss logical restore, see *z/OS DFSMSdss Storage Administration*.

Beginning in z/OS V2R1, zFS enhanced its support for the backup change activity flag in the VTOC (D1DSCHA in the Format 1/8). This flag indicates to a program (like DFSMShsm) whether the backup of a file system is needed (that is, data in the file system has been modified since the last backup).

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

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

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

Beginning in z/OS V2R1, zFS supplies an application programming interface that can be used to reset the change activity flag for a file system. This interface is intended to be used by DFSMSdss when doing a backup of a mounted zFS file system. For more information, see "Reset Backup Flag" on page 334.

Chapter 7. Migrating data from HFS or zFS to zFS

You can migrate data from HFS to zFS, or you might need to copy data efficiently from an existing zFS file system to a larger one, or to one that is created with different attributes (for example, if you want to have a secondary allocation to enable it to be dynamically grown). In all cases, the target file system can be version 1.4 as well as version 1.5. If all of the systems at your site are running at least version z/OS V2R1, then it is recommended that you use version 1.5 aggregates for your new file systems.

Guideline: Do not use the HFS to zFS migration tool if you are migrating your 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*.

Using the z/OS HFS to zFS migration tool

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

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

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

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

Using the z/OS UNIX pax command

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

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

Using an intermediate archive file

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

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

1. Move to the source (HFS) file system mounted at /etc/dfs
   ```
   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
   ```
   pax -wvf -o saveext /tmp/zfs1.pax .
   ```
3. Move to the target (zFS) file system mounted at /etc/dir1/testzfs1
   ```
   cd /etc/dir1/testzfs1
   ```
4. Read the archive file into the zFS file system mounted at /etc/dir1/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/dir1/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/dir1/testzfs1
   ```
   pax -rwvCMX -p -o saveext eW . /etc/dir1/testzfs1
   ```

Chapter 8. Performance and debugging

This section discusses performance tuning techniques and what should be done if a problem occurs that requires IBM service assistance. The examples are for illustrative purposes only.

In releases prior to z/OS V2R2, it was typical for the 4-byte counters used in the reports to wrap. In z/OS V2R2, 8-byte counters are used, which allows for monitoring of much longer time periods. The numbers being output into the report fields still use the same field width sizes, with the addition of a letter to indicate the units of the number if it is too large to fit into the field.

Letter	Unit of number
b	The number should be multiplied by 1,000,000,000.
G	The number should be multiplied by 1,073,741,824.
t	The number should be multiplied by 1000.
T	The number should be multiplied by 1,099,511,627,776.
tr	The number should be multiplied by 1,000,000,000,000.
m	The number should be multiplied by 1,000,000.
K	The number should be multiplied by 1024.
M	The number should be multiplied by 1,048,576.

Performance tuning

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

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

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

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

Total cache size

In releases prior to z/OS V2R2, the total storage size available for all the caches in the zFS address space had to be less than 2 GB. If the cache sizes specified in the

IOEFSPRM file were too large, zFS would terminate. 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, the total address space cache storage was restricted to approximately 1.5 GB. Use **`modify zfs,query,storage`** to determine the total allocated zFS storage. See "STOR" on page 85 for more information about determining how much of the available zFS address space storage is being used by the zFS caches.

In z/OS V2R2, zFS uses 64-bit storage above the 2 GB line. Therefore, zFS cache sizes are no longer restricted by the 2 GB storage size. Caches start at the minimum size during zFS initialization, and are allowed to grow as needed to the size specified in the IOEFSPRM file. Carefully consider how large you want your zFS caches to be, taking into account such things as the amount of real and auxiliary storage in your system.

You might see messages IOEZ00188A, IOEZ00662I, or IOEZ00663I, which indicate that the zFS address space is low on storage. If it continues to be a problem, consider using **`zfsadm config`** to dynamically decrease the size of caches that do not reside in data spaces, such as the metadata or vnode caches.

The **`modify zfs,query,all`** command also shows the total zFS storage that is allocated, but includes the storage that is allocated for all the caches and everything else zFS might need. The zFS address space caches include the following caches:
- "Metadata cache"
- "Vnode cache" on page 69
- "Log file cache" on page 70

The data in the user file cache is stored in data spaces, not zFS address space storage.

Metadata cache

The metadata cache is used to contain all file system metadata; this metadata includes all directory contents, file status information (such as, atime, mtime, size, and permission bits), and file system structures. Additionally, it also caches data for files smaller than 7 K. Essentially, zFS stores a file by using one of the following three methods. For more information about how zFS shows free blocks, see "Understanding zFS disk space allocation" on page 42.

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

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

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

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

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

The metadata cache is stored in the primary address space. Because the metadata cache contains only metadata and small files, it typically does not need to be nearly as large as the user file cache. The operator **modify zfs,query,all** command output shows statistics for the metadata cache including the cache hit ratio.

Vnode cache

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

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

|
User file cache

The user file cache is used to cache all “regular” files. It caches any file, no matter what its size, and performs write-behind and asynchronous read-ahead for files. It performs I/O for all files that are 7 KB or larger. For files smaller than 7 KB, I/O is typically performed through the metadata cache.

| The user file cache is allocated in data spaces. The default size of `user_cache_size`
| is calculated. For more information, see “IOEFSPRM” on page 220. However, you
| can tailor this size to meet your performance needs, based on your overall system memory. The maximum size for `user_cache_size` is 65,536 MB (64 GB). The general rule for any cache is to ensure a good hit ratio. Additionally, it is good to have a user file cache that is large enough for write-behind activity to occur. If the cache is too small, you need to recycle buffers more frequently and that might degrade write-behind performance. The MODIFY ZFS,QUERY,ALL command output shows the cache hit ratio, which is actually the “fault ratio”. To get the hit ratio, subtract the fault ratio from 100%.

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

Log files

Every zFS aggregate contains a log file that is used to record transactions that describe changes to the file system structure. This log file is, by default, 1% of the aggregate size; but, you can tailor it on the **ioeagfmt** 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 4 KB buffers used to contain log file updates. You must not modify the log file cache size unless under the direction of IBM service. Log file buffers are always written asynchronously to disk and typically need to be waited upon only when the log is becoming full, or if a file is in file synchronization (fsync).

The log file cache is stored in the primary address space and its default size is 16 MB. The log file cache is grown dynamically by adding two 4 KB buffers for each attached aggregate. This growth ensures that each aggregate always has one one log cache 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.

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 option ensures that the cache experiences no paging and avoids page fixing for each I/O. This option does come at the expense of using real storage for the cache, which means the real storage is not available for other applications.

If your file system performance is critical and you have enough real memory to support it, the `fixed` option can be useful. Otherwise, do not set it.

I/O balancing

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

zFS MODIFY ZFS,QUERY,ALL operator commands also provide reports that show I/O rates per aggregate, and file system request rates per aggregate and per file system. This information, along with DASD performance information from RMF or performance products similar to RMF can be used to balance I/O among your

DASD. For example, you can use the **query** command output to show the file systems that can be moved to different DASD to achieve a better balance among disks.

Monitoring zFS performance

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

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

Note: If zFS is running in the OMVS address space, the syntax of the modify command is as follows:

```
modify omvs,pfs=zfs,query,<report>,<option>
```

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

CTKC Displays the client token manager statistics. CTKC is only present when the system is a sysplex client of another system and the zFS CTKC component on this system sent a message to another system. See "CTKC" on page 72 for details of the report.

DATASET
Displays zFS statistics about file systems.

FILE Provides a detailed breakdown of requests per zFS file system and aggregate. By default, this report lists only file systems and aggregates that had active requests since the last statistics reset. If you use the ALL option, you get all file system and aggregates regardless of whether they were active or not. See "FILE" on page 74 for details of the report.

IOBYDASD
Displays the I/O statistics by currently attached DASD volumes including the total number of waits for I/O and the average wait time per I/O. See "IOBYDASD" on page 75 for details of the report.

KN Provides counts of calls that are made to zFS from z/OS UNIX and the average response time of each call. This information is the basic measure of zFS performance. See "KN" on page 76 for details of the report.

LFS Provides detailed file system statistics including the performance of the zFS metadata cache, the vnode cache, and the aggregate I/O statistics. See "LFS" on page 78 for details of the report.

LOCK Provides a measure of lock contention and how often z/OS UNIX threads wait for certain events such as user file cache reclaim. See "LOCK" on page 81 for details of the report.

LOG Provides performance information for the log file cache. See "LOG" on page 82 for details of the report.

STKM
Displays the current server token manager (STKM) statistics. See "STKM" on page 83 for details of the report.

STOR Provides a detailed breakdown of zFS allocated storage by component. By default, this report lists only storage usage by zFS component. If you use the DETAILS option, you get more detailed information for each zFS component. See "STOR" on page 85 for details of the report.

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

VM Provides performance information for the user file cache including cache hit ratios, I/O rates, and storage usage. See "VM" on page 89 for details of the report.

Resetting performance monitoring data

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

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

Note: If zFS is running in the OMVS address space, the syntax of the modify command is:

```
modify omvs,pfs=zfs,reset,<report>
```

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

To start the monitoring period at 1 PM, enter `MODIFY ZFS,RESET,ALL`.

To end the monitoring period at 3 PM, enter `MODIFY ZFS,QUERY,ALL`.

Sample zFS QUERY reports

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

- "CTKC"
- "DATASET" on page 73
- "FILE" on page 74
- "IOBYDASD" on page 75
- "KN" on page 76
- "LFS" on page 78
- "LOCK" on page 81
- "STKM" on page 83
- "STOR" on page 85
- "SVI" on page 88
- "VM" on page 89

CTKC

Displays the statistics relating to calls made to other systems caused by operations on the local system (called client operations). The output is displayed only when the system is a sysplex client of another system and the zFS CTKC component on this system has sent a message to another system. Figure 28 on page 73 shows an example of the total number of call counts and the average response time in milliseconds of the call to the system indicated (in this case NP1).

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

```
          SVI Calls to System NP1
          ----------------------------
SVI Call                     Count      Avg. Time
--------------------       ----------   ----------
GetToken                     211324       15.996
GetMultTokens                     0        0.000
ReturnTokens                     31        0.621
ReturnFileTokens                  0        0.000
FetchData                         0        0.000
StoreData                     27005        3.354
Setattr                      184762        4.486
FetchDir                         25       20.464
Lookup                           30        4.772
GetTokensDirSearch                0        0.000
Create                            3       17.921
Remove                            0        0.000
Rename                            0        0.000
Link                              0        0.000
ReadLink                          0        0.000
SetACL                            0        0.000
Statfs                           42        2.006
TSR                               0        0.000
FilesysSyncTable                  0        0.000
FileSyncMeta                      0        0.000
BitmapReserve                     0        0.000
BitmapUnreserve                   0        0.000
BitmapReclaim                     0        0.000
FileUpdateIB                      0        0.000
FileCreateIB                      0        0.000
FwdReaddir                        0        0.000
LkupInvalidate                    0        0.000
FileDebug                         0        0.000
FetchPage                         0        0.000
ServerIO                          0        0.000
BulkFetchStatus                   0        0.000
Convert                           0        0.000
ConvertFID                        0        0.000
  --------------------       ----------   ----------
*TOTALS*                     423222       10.162
```

Figure 28. Sample CTKC report

DATASET

The DATASET report, shown in Figure 29, lists zFS data set statistics. Table 2 on page 74 describes the contents of the report.

```
Printing Dataset Allocation Stats
        Allocates          2
        Allocates failed   0
        Unallocates        2
        Unallocates failed 0
        Opens              2
        Open failures      0
        Closes             2
```

Figure 29. Sample DATASET report

Table 2. DATASET report fields

Field name	Contents
Allocates	Number of allocations issued by zFS for zFS data sets.
Allocates failed	Number of allocations issued by zFS for zFS data sets that were unsuccessful.
Unallocates	Number of unallocations issued by zFS for zFS data sets.
Unallocates failed	Number of unallocations issued by zFS for zFS data sets that were unsuccessful.
Opens	Number of opens issued by zFS for zFS data sets.
Opens failed	Number of opens issued by zFS for zFS data sets that were unsuccessful.
Closes	Number of closes issued by zFS for zFS data sets.

FILE

The FILE report, which is shown in Figure 30, lists every file system that was active since the last reset by default. If you use the `ALL` option, it lists all file systems. The file systems are listed in the report with the most active file systems listed first. Table 3 describes the contents of the report.

```
FILE:
File System Name                             Aggr #  Flg  Operations
---------------------------------------------  ------  ---  ----------

OMVS.ZFS.DFBLD.DFSSRC                              8  AM       274472

OMVS.ZFS.LOCAL                                     9  AM       111722

OMVS.ZFS.DCEDFBLD.DCES390.ETC.DCE                 10  AMQ       81632

OMVS.ZFS.DCEDFBLD.DFSLOCAL                        12  AM        52154

OMVS.ZFS.DCEDFBLD.OS390R10.ETC                     4  AMC       44108

OMVS.ZFS.GPLTOOLS                                  6  AM         8458

OMVS.ZFS.BLDTOOLS                                  7  AM         8120

OMVS.ZFS.DCEDFBLD.VAR                              5  AM          314

OMVS.ZFS.USR.LOCAL                                11  AM           54
```

Figure 30. Sample FILE report

Table 3. FILE report fields

Field name	Contents
Aggr #	The aggregate ID that can be seen in the **zfsadm lsfs** -long command.

Table 3. FILE report fields (continued)

Field name	Contents
Flg	Indicates the aggregate status, as follows: **A** Attached **G** Growing **L** Locally owned **M** Mounted **O** Offline (disabled) **Q** Quiesced **S** Sysplex-aware (if the aggregate is sysplex-aware for read/write) This command only reports on locally mounted (attached) aggregates. You can use the operator ROUTE command to issue this command to all systems in your sysplex (for example, ROUTE *ALL,F ZFS,QUERY,FILE,ALL). Note that the zFS owning system can flag an aggregate as growing (G) while the other (zFS client) systems can flag it as quiesced (Q). That flagging occurs because an aggregate that is growing is quiesced on all other systems.
Operations	Indicates the count of z/OS UNIX vnode calls to that particular file system; it is not an I/O rate. You can use the RMF DASD reports, the LFS Aggregate I/O report, and the FILE report to balance your file systems among disks to provide a more even I/O spread.

IOBYDASD

As Figure 31 shows, the IOBYDASD report lists the currently attached DASD by volume. This report is important for viewing the average wait time per I/O (in milliseconds).

```
IOEZ00438I Starting Query Command IOBYDASD.
                     zFS I/O by Currently Attached DASD/VOLs

DASD    PAV
VOLSER  IOs    Reads  bytes   Writes bytes   Waits    Average Wait
------  -----  -----  -----   -----  -----   -----    ------------
CFC002      1     5m    40M      2m    52M      5m           5.964
SMBD80      1   5136  21784    197t     1M    138t           3.377
ZFSD50      1     3m    27M      1m    32M      4m           7.629
ZFSD32      1   5097  21620   57227     1M   13173           4.372
ZFSD33      1     4m    33M      2m    37M      5m           8.316
ZFS183      1   663t     4M    262t     4M    669t           8.506

Total number of waits for I/O:    16111355
Average wait time per I/O:           7.228
```

Figure 31. Sample IOBYDASD report

Table 4 describes the contents of the report.

Table 4. IOBYDASD report fields

Field name	Contents
DASD VOLSER	The DASD volumes that contain the zFS aggregates.
PAV IOs	The maximum number of concurrent I/O requests to volume.
Reads	The number of read I/O requests.
K bytes	The number of bytes read or written in K units.
Writes	The number of write I/O requests.

Table 4. IOBYDASD report fields (continued)

Field name	Contents
Waits	The number of waits for I/O completion.
Average Wait	The average wait time for I/O requests in milliseconds.
Total number of waits for I/O	Total of Waits column
Average wait time per I/O	The average of the Average Wait times, in milliseconds.

KN

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

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

If you see abnormally long times that are listed for `zfs_lookup`, `zfs_creates`, or `zfs_removes` and you are using v4 directories, you might have a zFS large directory problem. For information about the zFS large directory performance problem, see "Minimum and maximum file system sizes" on page 46.

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

```
F ZFS,QUERY,KNPFS
IOEZ00438I Starting Query Command KN. 525
          PFS Calls on Owner
          --------------------
Operation              Count  XCF req.    Avg Time
---------         ----------  ----------  ----------
zfs_opens               9456           0       1.778
zfs_closes              9444           2       0.788
zfs_reads             123636           0       0.145
zfs_writes            109510           0       2.475
zfs_ioctls                66           0       0.685
zfs_fileinfos              0           0       0.000
zfs_converts               0           0       0.000
zfs_getattrs           17561           1       0.033
zfs_setattrs             699           0       0.377
zfs_accesses             709           0       0.099
zfs_lookups            52931         959       0.396
zfs_creates             3158         780       4.659
```

```
    zfs_removes              633           0       4.164
    zfs_links                 16           0       0.978
    zfs_renames              640           0       2.108
    zfs_mkdirs               780         164       4.346
    zfs_rmdirs               378           0       4.731
    zfs_readdirs             440          23       5.871
    zfs_symlinks              59          19       4.393
    zfs_readlinks            132           0       0.076
    zfs_fsyncs               188           0       9.310
|   zfs_inactives           2347           5       0.013
|   zfs_setacls                0           0       0.000
|   zfs_getacls                0           0       0.000
    zfs_truncs               553           0       3.278
    zfs_recoveries             0           0       0.000
    zfs_audits               357           0       0.051
    zfs_pfsctls               85          13       6.528
    zfs_statfss              245           0       0.278
    zfs_vgets                  0           0       0.000
    zfs_mounts                22           0     435.707
    zfs_unmounts               0           0       0.000
    zfs_vinacts                0           0       0.000
|   zfs_sync                   0           0       0.000
    ---------         ----------  ----------  ----------
    *TOTALS*              334045        1966       1.125
    IOEZ00438I Starting Query Command KN. 526
              PFS Calls on Client
              --------------------
    Operation              Count  XCF req.    Avg Time
    ---------         ----------  ----------  ----------
    zfs_opens               6373         121       2.665
    zfs_closes              6365        2675       2.243
    zfs_reads              17402        2013       0.636
    zfs_writes             12043        8904      10.122
    zfs_ioctls                15           0       0.096
|   zfs_fileinfos              0           0       0.000
|   zfs_converts               0           0       0.000
    zfs_getattrs           14560        3972       1.170
    zfs_setattrs            1183          17       0.102
    zfs_accesses             194           0       0.331
    zfs_lookups            47122        1606       0.340
    zfs_creates             2492        2491       8.749
    zfs_removes              485         485       6.049
    zfs_links                 15          15       4.582
    zfs_renames             1151        1151       5.673
    zfs_mkdirs               603         603       6.548
    zfs_rmdirs               582         582       5.834
    zfs_readdirs             317          27       3.625
    zfs_symlinks              55          55       8.031
    zfs_readlinks            115          72       1.854
    zfs_fsyncs                 0           0       0.000
|   zfs_inactives           2077           0       0.016
|   zfs_setacls                0           0       0.000
|   zfs_getacls                0           0       0.000
    zfs_truncs                 0           0       0.000
    zfs_recoveries             0           0       0.000
    zfs_audits                92           0       0.021
    zfs_pfsctls                0           0       0.000
    zfs_statfss               50          50       4.741
    zfs_vgets                  0           0       0.000
    zfs_mounts                 7           0     313.115
    zfs_unmounts               0           0       0.000
    zfs_vinacts                0           0       0.000
|   zfs_sync                   0           0       0.000
    ---------         ----------  ----------  ----------
    *TOTALS*              113298       24839       2.122
```

```
IOEZ00025I zFS kernel: MODIFY command - QUERY,KNPFS completed successfully.
```

LFS

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

```
F ZFS,QUERY,LFS
IOEZ00438I Starting Query Command LFS. 421
                     zFS Vnode Op Counts

Vnode Op              Count    Vnode Op              Count
----------------- ----------    ----------------- ----------
efs_hold                  0    efs_readdir           67997
efs_rele                  0    efs_create          1569039
efs_inactive              0    efs_remove          1945874
efsvn_getattr       9856523    efs_rename           235320
efs_setattr              40    efs_mkdir            237359
efs_access          1656502    efs_rmdir            238004
efs_lookup         21545682    efs_link             237318
efs_getvolume             0    efs_symlink          237318
efs_getlength             0    efs_readlink              0
efs_afsfid                0    efs_rdwr                  0
efs_fid                   0    efs_fsync                 0
efs_vmread                0    efs_waitIO                9
efs_vmwrite               0    efs_cancelIO              0
efs_clrsetid              0    efs_audit              5425
efs_getanode          16640    efs_vmblkinfo             0
efs_readdir_raw           0    efs_convert               0

Average number of names per convert                      0
Number of version5 directory splits                    126
Number of version5 directory merges                     63

Total zFS Vnode Ops                               37849050

                        zFS Vnode Cache Statistics

Vnodes      Requests     Hits    Ratio  Allocates   Deletes
  ---------- ---------- ---------- ----- ---------- ----------
  200000   25908218   22431383  86.580%         0         1

zFS Vnode structure size: 224 bytes
zFS extended vnodes: 200000, extension size 816 bytes (minimum)
Held zFS vnodes:        2914 (high      29002)
Open zFS vnodes:           0 (high         10)
Reusable:             197085

Total osi_getvnode Calls:    3886774 (high resp          0) Avg. Call
Time:         0.069 (msecs)
Total SAF Calls:           11050540 (high resp          1) Avg. Call
Time:         0.008 (msecs)

Remote Vnode Extension Cleans              0
                        zFS Fast Lookup Statistics

Buffers     Lookups      Hits    Ratio  Neg. Hits   Updates
---------- ---------- ---------- ----- ---------- ----------
      1000          0          0   0.0%          0          0

                         Metadata Caching Statistics

Buffers   (K bytes)  Requests     Hits    Ratio   Updates   PartialWrt
--------- --------- ---------- ---------- ------ ---------- ----------
    32768    262144   77813570   77529130  99.6%   27943073     423524
```

```
                  I/O Summary By Type
                -------------------

Count       Waits       Cancels     Merges      Type
----------  ----------  ----------  ----------  ----------
     33006        7701           0           0  File System Metadata
    680516        1020           0       56366  Log File
        11           1           0           0  User File Data

                  I/O Summary By Circumstance
                  ---------------------------

Count       Waits       Cancels     Merges      Circumstance
----------  ----------  ----------  ----------  ------------
      7213        6553           0           0  Metadata cache read
         1           1           0           0  User file cache direct read
         4           4           0           0  Log file read
         0           0           0           0  Metadata cache async delete write
         0           0           0           0  Metadata cache async write
         0           0           0           0  Metadata cache lazy write
         0           0           0           0  Metadata cache sync delete write
         0           0           0           0  Metadata cache sync write
        10           0           0           0  User File cache direct write
         1           1           0           0  Metadata cache file sync write
     16981         861           0           0  Metadata cache sync daemon write
         0           0           0           0  Metadata cache aggregate detach write
         0           0           0           0  Metadata cache buffer block reclaim write
         0           0           0           0  Metadata cache buffer allocation write
         0           0           0           0  Metadata cache file system quiesce write
      8811         286           0           0  Metadata cache log file full write
    680512        1016           0       56366  Log file write
         0           0           0           0  Metadata cache shutdown write
         0           0           0           0  Format, grow write

                     zFS I/O by Currently Attached Aggregate

DASD   PAV
VOLSER IOs Mode  Reads       K bytes     Writes      K bytes
Dataset Name
------ --- ----  ----------  ----------  ----------  ----------
ZFSAGGR.BIGZFS.DHH.FS14.EXTATTR
ZFSD18   1  R/W          44         344        1831       17224
ZFSAGGR.BIGZFS.DHH.FS1.EXTATTR
ZFS121   1  R/W        6509       52056      648750    10276788

------           ----------  ----------  ----------  ----------
*TOTALS*
     2                 6553       52400      650581    10294012


Total number of waits for I/O:       8722
Average I/O wait time:               115.334 (msecs)
IOEZ00025I zFS kernel: MODIFY command - QUERY,LFS completed successfully
```

Table 5. LFS report sections

Field name	Contents
zFS Vnode Op Counts:	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.

Table 5. LFS report sections (continued)

Field name	Contents
zFS Vnode Cache Statistics:	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. **Held zFS vnodes** is the number of vnodes that z/OS UNIX has required of zFS to currently access. **high** is the largest number of vnodes that z/OS UNIX required of zFS to access at one time (during a peak time). z/OS UNIX also determines when files are to be opened and closed. **Open zFS vnodes** is the number of vnodes that represent currently open files. **high** is the largest number of files open at the same time. Generally, a good hit ratio for this cache is preferable because a miss means initializing the data structures and initialization requires a read of the object's status from disk. Often this is in the metadata cache, but it is not guaranteed. Consequently a vnode cache lookup miss might require an I/O wait. The vnode structure size is shown; however, additional data structures anchored from the vnode also take space. Everything added together yields over 1 K of storage per vnode. Consider this when planning the size of this cache. Also note that initializing a vnode will not require an I/O if the object's status information is in the metadata cache, thus a good size metadata cache can be as useful—often more useful than an extremely large vnode cache. `Total osi_getvnode Calls` is the number of times zFS called the osi_getvnode interface of z/OS UNIX to get a z/OS UNIX vnode to correspond to a new zFS vnode. Its `high resp` is the number of calls that took longer than a second to complete. `Avg. Call Time` is the average number of milliseconds each call took to complete. `Total SAF Calls` is the number of calls zFS made to the security product via the SAF interface. `high resp` is the number of these security calls that took longer than a second to complete. `Avg. Call Time` is the average number of milliseconds each call took to complete.
zFS Fast Lookup Statistics:	Shows the basic performance characteristics of the zFS fast lookup cache. The fast lookup cache is used on the owning system for a zFS sysplex-aware file system to improve the performance of the lookup operation. There are no externals for this cache (other than this display). The statistics show the total number of buffers (each are 8K in size), the total number of lookups, the cache hits for lookups and the hit ratio. The higher the hit ratio, the better the performance.
Metadata Caching Statistics:	Shows the basic performance characteristics of the metadata cache. The metadata cache contains a cache of all disk blocks that contain metadata and any file data for files less than 7 K in size. For files smaller than 7 K, zFS places multiple files in one disk block (for zFS a disk block is 8 K bytes). Only the lower metadata management layers have the block fragmentation information, so the user file I/O for small files is performed directly through this cache rather than the user file cache. The statistics show the total number of buffers (each are 8 K in size), the total bytes, the request rates, hit ratio of the cache, Updates (the number of times an update was made to a metadata block), and Partial writes (the number of times that only half of an 8-K metadata block needed to be written). The higher the hit ratio the better the performance. Metadata is accessed frequently in zFS and all metadata is contained only (for the most part) in the metadata cache therefore, a hit ratio of 80% or more is typically sufficient.

Table 5. LFS report sections (continued)

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

LOCK

The LOCK report is mainly for IBM service to use when diagnosing performance problems relating to lock contention. This report shows a detailed breakdown of how often zFS waits for locks. It also shows which locks cause the most contention. Additionally, the report monitors how often a thread sleeps while waiting for an event. Figure 32 on page 82 shows an example of a report.

```
LOCK:
                                     Locking Statistics


Untimed sleeps:        22   Timed Sleeps:         0   Wakeups:        21

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

Total monitored sleeps:                  22
Average monitored sleep time:             0.792 (msecs)

Total starved waiters:                    0
Total task priority boosts:               0

                Top 15 Most Highly Contended Locks
  Thread      Async       Spin
    Wait      Disp.       Resol.        Pct.    Description
----------  ----------  ----------      -----   --------------
       877           0         899    35.763%   Log system map lock
      1464           0          40    30.285%   Anode bitmap allocation handle
       481           0          28    10.249%   Anode fileset quota lock
       291           0          42     6.705%   Transaction lock
       205           0          62     5.376%   Metadata-cache buffer lock
       210           0           4     4.309%   Anode fileset handle lock
        84          68           7     3.201%   User file cache main segment lo
         0          55           0     1.107%   Volser I/O queue lock
        38           0           0     0.765%   Vnode-cache access lock
         2          23          11     0.724%   Transaction-cache main lock
        19           0           3     0.443%   Transaction-cache equivalence c
        21           0           0     0.422%   Async IO event lock
         0          14           0     0.281%   Cache Services association main
         6           0           0     0.120%   Cache Services hashtable resize
         0           0           5     0.100%   Transaction-cache complete list

Total lock contention of all kinds:      4966
                Top 15 Most Common Thread Sleeps
Thread Wait     Pct.        Description
-----------     -------     -----------
         22    100.000%     Transaction allocation wait
          0      0.000%     OSI cache item cleanup wait
          0      0.000%     Directory Cache Buffer Wait
          0      0.000%     User file cache Page Wait
          0      0.000%     User file cache File Wait
```

Figure 32. Example of a LOCK report

LOG

Figure 33 on page 83 shows performance statistics for the Log File Cache. The Log File Cache is a write-only cache that is stored in the primary address space and is shared among all attached R/W aggregates that are zFS-owned on a system. Because zFS will ensure that there is at least one Log File Buffer for each aggregate it represents, modifying IOEFSPRM configuration option `log_cache_size` to change the size of the cache should not be necessary.

An example of a query of log cache statistics report is provided. Each field in the report is self-explanatory. This information is useful only to IBM service personnel, but is shown here for completeness.

```
                         Log File Caching Statistics

Logs
--------
         7 : Log files cached
         0 : Log file recoveries performed
      1494 : Log file syncs (filesys quiesce)

Policies
--------
        16 : Reclaim pct. (amount reclaimed at log-full time)
        16 : Maximum log pages per IO
        50 : Inactive buffer schedule pct. (of log size)

Storage
--------
      4116 : Log Cache Size (in 4K pages, fixed=NO)
         0 : Pct. of cache in-use
         0 : Free page obtain waits
         0 : Allocations to avoid deadlock

Transactions
------------
 148034756 : Transactions started
  42074853 : Transactions merged
      98.1 : Average number of transactions batched together
    324426 : Sync calls to an active transaction
   1059260 : Sync calls to a completed transaction

IOs and Blocks
--------------
         0 : Log IOs in progress
     10403 : Dirty metadata blocks
    893555 : Metadata block kill calls
   1507583 : Log File writes initiated
       5.2 : Average number of pages per log write
       719 : Avoided IOs for metadata block due to deallocation
    234215 : Scheduled not-recently-updated (NRU) metadata blocks
      16.4 : Average number of blocks per NRU IO
    848508 : Metadata buffers forced to disk
       0.9 : Avg where metadata write forced write of log
      99.8 : Pct. of metadata buffer forces waited on log IO
      3250 : Log-full processing calls
     262.4 : Avg number of metadata blocks written per log-full

Update Records
--------------
     330.2 : Avg number of update records per log IO.
  13709331 : Number of NBS records written
1514937445 : Number of metadata buffer updates
   3814761 : Number of updates requiring old-byte copying
 796990391 : Avoided buffer update records due to overlap
   2854045 : Avoided merge update records due to overlap
```

Figure 33. Sample LOG report

STKM

The STKM report, as shown in Figure 34 on page 84, lists the server token manager statistics. LOCALUSR is the local system (the server). ZEROLINK is a “special client” used to handle zero link count files and vnode inactivations. Table 6 on page 84 describes the contents of the report.

```
                 Server Token Manager (STKM) Statistics
                 --------------------------------------
Maximum tokens:        30724       Allocated tokens:       30720
Tokens In Use:         27687       File structures:        27696
Token obtains:       3542592       Token returns:        3485439
Token revokes:       1309562       Async Grants:               0
Garbage Collects:        666       Thrash Resolutions:         0
Thrashing Files:           8

                          Usage Per System:
System    Tokens      Obtains    Returns     Revokes      Async Grt
--------  ----------  ----------  ----------  ----------  ----------
     NP1        3781      897812      894887      502842           0
     NP2       15147     1233561     1188354      415917           0
     NP3           3         912         909           0           0
     NP4        8756     1410737     1402062      504757           0
ZEROLINK           0           0           0           0           0
LOCALUSR           0           0           0           0           0

             Thrashing Objects:
Inode      Uniquifier   File system
---------- ---------- --------------------
     19305     181700 PLEX.ZFS.SMALL2
         1          1 ZFSAGGR.BIGZFS.DHH.FS4.EXTATTR
       711     184733 PLEX.ZFS.SMALL2
         1          1 ZFSAGGR.BIGZFS.DHH.FS14.EXTATTR
         1          1 ZFSAGGR.BIGZFS.DHH.FS1.EXTATTR
        13          1 ZFSAGGR.BIGZFS.DHH.FS4.EXTATTR
        11          1 ZFSAGGR.BIGZFS.DHH.FS14.EXTATTR
     21761       8528 ZFSAGGR.BIGZFS.DHH.FS6.EXTATTR
```

Figure 34. Sample STKM report

Table 6. STKM report fields

Field	Contents
Maximum tokens:	Lists the token limit at the server which is defined by the IOEFSPRM configuration option `token_cache_size`. The server runs garbage collection to ensure that token maximum is not exceeded. In some cases, the system workload might cause the token maximum to be exceeded, such as when there are many open files.
Allocated tokens:	Number of tokens allocated in server memory. Tokens are allocated as needed, up to *maximum* tokens.
File structures:	Number of file structures.
Tokens In Use:	Number of tokens currently held by all clients and the local system. If this number approaches maximum tokens, then consider increasing the `token_cache_size` setting.
Token obtains:	Total number of token obtains by all clients and local system.
Token revokes:	Total number of token revokes by all clients and local system.
Token returns:	Total number of token returns by all clients and local system.
Async grants:	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:	Number of garbage collections of tokens. Garbage collection is used to keep the total number of client/local system tokens below the maximum whenever possible. If this number gets high, consider increasing the `token_cache_size` setting.
Thrashing files:	Number of files or directories that are thrashing.
Thrashing resolutions:	Number of trashing situations resolved.

The report indicates how many tokens each system currently has, how many token obtains and token returns each system has done, and how many times each system has had some tokens revoked.

The report also contains a list of objects that are undergoing thrashing. *Thrashing* means that the system that owns the file system containing the object needed to keep revoking tokens for the object because multiple systems were repeatedly writing to it. The list contains the inode and uniquifier of the object and the file system that contains it.

STOR

The STOR report shows the storage that zFS has allocated below the 2 G addressing line, and the storage that is allocated above the 2 G address line. The STOR report also provides a breakdown of zFS storage usage. This report can be used to determine how much storage zFS uses, based on a configuration change (such as increasing or decreasing a zFS cache through the **`zfsadm config`** command). Figure 35 on page 86 shows a sample report and Table 7 on page 88 explains the contents of each field. (Not shown here is the output of QUERY,STOR,DETAILS, which breaks down each component and shows how much storage is used for each data structure class; this report is intended primarily for IBM service.)

You can check zFS storage usage by issuing the operator command MODIFY ZFS,QUERY,STORAGE. If you compare the third line of data (`USS/External Storage Access Limit`) to the fourth line (Total Storage Below 2G Bar Allocated), you can determine how close zFS is to using its maximum storage below the 2 G addressing line. The vast majority of the storage used by zFS should be above the 2 G addressing line. The storage allocated below the 2 G Bar should be far less than the USS/External Storage Access Limit. For example, in figure, the storage allocated below the 2 G bar (approximately 231 M) is much less than the USS/External storage access limit (1793 M).

If the Total Storage Below 2G Bar Allocated becomes greater than or equal to the USS/External Storage Access Limit, zFS issues message IOEZ00662I. If the Total Storage Below 2G Bar Allocated approaches the value of the USS/External Storage Access Limit, you can attempt to dynamically decrease the caches using the **`zfsadm config`** command. (Also make the corresponding changes in your IOEFSPRM file for the next zFS restart.) Alternatively, you can stop and restart zFS after making cache size changes to your IOEFSPRM file.

If zFS failed to initialize and is not active, decrease some of your zFS IOEFSPRM settings, especially if they are significantly larger than the default values, and restart zFS. The settings to review include:
- `meta_cache_size`
- `recovery_max_storage`
- `token_cache_size`
- `vnode_cache_size`

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

You can also use the MODIFY ZFS,QUERY,STORAGE command to see Total Storage Above 2G Bar Allocated.. If the amount of storage allocated becomes more than desired, overall system performance can be impacted. If this occurs, you can attempt to use the **zfsadm config** command to decrease the size of a zFS cache that is using too much storage dynamically.

In the report, Discarded (or unbacked) storage is storage that is allocated to zFS, but is currently not in use. So, it is not occupying real storage frames, which reduces the need for paging by the system. If the storage is needed later, then it will again be used.

```
IOEZ00438I Starting Query Command STORAGE.
                 zFS Primary Address Space <2G Stge Usage
                 ----------------------------------------

Total Storage Below 2G Bar Available:  1943011328
Non-critical Storage Limit:            1922039808
USS/External Storage Access Limit:     1880096768
Total Storage Below 2G Bar Allocated:   242671616

IOEFSCM Heap Bytes Allocated:            26560184
IOEFSCM Heap Pieces Allocated:             1671
IOEFSCM Heap Allocation Requests:          1680
IOEFSCM Heap Free Requests:                   9

IOEFSKN Heap Bytes Allocated:             3610517
IOEFSKN Heap Pieces Allocated:           54383
IOEFSKN Heap Allocation Requests:       242678
IOEFSKN Heap Free Requests:             188295

        Storage Usage By Sub-component
 Bytes              No. of  No. of
 Allocated   Pieces Allocs  Frees   Component
 ----------  ------ ------  ------  ---------
       2375       7      7       0  Interface
      14544       2      2       0  Media Manager I/O driver
       1888       5      5       0  Trace Facility
     434088       7      7       0  Message Service
     546428     164    164       0  Miscellaneous
      33168       1      1       0  Aggregate Management
     200384       2      2       0  Filesystem Management
      32160      27     36       9  Administration Command Handling
       1264       5 130652  130647  Vnode Management
      50632      14  57614   57600  Anode Management
          0       0      0       0  Directory Management
       1904       2      2       0  Log File Management
        272       1      1       0  Metadata Cache
          0       0      0       0  Transaction Management
       2192       1      1       0  Asynchronous I/O Component
     119436    1909   1909       0  Lock Facility
      10440     348    348       0  Threading Services
    1768592   51561  51597      36  Cache Services
      49366       8      9       1  Config. parameters processing
       8496       4      4       0  User File Cache
```

Figure 35. Sample STOR report (part 1 of 2)

```
    313784     182     182       0  Storage Management
     12456     126     128       2  XCF Services
         0       0       0       0  Cross system attach validation
      5464       4       4       0  Server Token Manager (STKM)
       224       1       1       0  Server Token Cache (STKC)
       936       1       1       0  Client Token Cache (CTKC)
         0       0       0       0  Server Vnode Interface (SVI)
         0       0       0       0  Name Space (NS)
        24       1       1       0  Directory storage
         0       0       0       0  Salvage storage
IOEZ00438I Starting Query Command STORAGE.
                 zFS Primary Address Space >2G Stge Usage
                 ----------------------------------------

Total Storage Above 2G Bar Available:          4294963200M
Total Storage Above 2G Bar Allocated:           1766850560

Total Bytes Allocated by IOEFSCM (Stack+Heap):    22020096
IOEFSCM Heap Bytes Allocated:                     22020096
IOEFSCM Heap Pieces Allocated:            462
IOEFSCM Heap Allocation Requests:         462
IOEFSCM Heap Free Requests:                 0

Total Bytes Allocated by IOEFSKN (Stack+Heap):   648019968
Total Bytes Discarded (unbacked) by IOEFSKN:      55504896
IOEFSKN Heap Bytes Allocated:                    546676397
IOEFSKN Heap Pieces Allocated:        1122125
IOEFSKN Heap Allocation Requests:     6739163
IOEFSKN Heap Free Requests:           5617038

       Storage Usage by Sub-component
   Bytes            No. of  No. of
 Allocated   Pieces  Allocs  Frees   Component
 ----------  ------  ------  ------  ---------
    459628      16      16       0  Interface
    675080     193     213      20  Media Manager I/O driver
  73400320       2       2       0  Trace Facility
         0       0       0       0  Message Service
   8399061     284     315      31  Miscellaneous
     77216     117     126       9  Aggregate Management
     21376      14      14       0  Filesystem Management
      1464      10      20      10  Administration Command Handling
  15026992   56535  453053  396518  Vnode Management
  43586724  329845  387711   57866  Anode Management
         0       0       0       0  Directory Management
  45070848   44098  267949  223851  Log File Management
 164305040   38354   38366      12  Metadata Cache
         0       0       0       0  Transaction Management
   5874464   68159   69176    1017  Asynchronous I/O Component
   1048576       1       3       2  Lock Facility
   1048576       1       1       0  Threading Services
  87901088  490273  1214627  724354  Cache Services
         0       0       0       0  Config. parameters processing
   4696016   16004   16022      18  User File Cache
   6047280    4322    4607     285  Storage Management
  65608048    1678    1678       0  XCF Services
     17680      13      22       9  Cross system attach validation
   1167992    6050  4117454  4111404  Server Token Manager (STKM)
    263528    3058    3058       0  Server Token Cache (STKC)
  20930824   63097   63097       0  Client Token Cache (CTKC)
         0       0  101623  101623  Server Vnode Interface (SVI)
         0       0       9       9  Name Space (NS)
   1048576       1       1       0  Directory storage
         0       0       0       0  Salvage storage
```

Figure 36. Sample STOR report (part 2 of 2)

Table 7. STOR report fields

Field name	Contents
Total storage below 2G bar available Total storage above 2G bar available	Total virtual storage in the zFS address space that is available for usage (such as caches, control blocks, and stacks).
Non-critical Storage Limit	The value that, when exceeded, will cause zFS to issue message `IOEZ00663I ZFS is critically low on storage`.
USS/External Storage Access Limit	The value that, when exceeded, will cause zFS to issue message `IOEZ00662I ZFS is low on storage`.
Total storage below 2G bar allocated Total storage above 2G bar allocated	The current usage of virtual storage in the zFS address space (requested by zFS and other components running in the zFS address space).
IOEFSCM Heap Bytes Allocated IOEFSKN Heap Bytes Allocated	The current amount of storage that is allocated to the zFS heaps.
IOEFSCM Heap Pieces Allocated IOEFSKN Heap Pieces Allocated	The current number of storage pieces that are in the IOEFSCM and IOEFSKN heaps.
Total Bytes Allocated by IOEFSCM (Stack + Heap) Total Bytes Allocated by IOEFSKN (Stack + Heap)	The total bytes of storage allocated by the zFS IOEFSCM and IOEFSKN components.
IOEFSCM Heap Allocation Requests IOEFSKN Heap Allocation Requests	Number of requests that zFS made to obtain heap storage since the last zFS storage statistics reset.
IOEFSCM Heap Free Allocated IOEFSKN Heap Free Allocated	Number of requests that zFS made to free heap storage since the last zFS storage statistics reset.
Storage Usage by Sub-component	Storage usage for each zFS component.
Total Bytes Discarded (unbacked) by IOEFSKN	Total number of bytes that IOEFSKN has discarded (made unbacked) from allocated storage.

SVI

The server vnode interface component handles this call. The example report that is shown in Figure 37 on page 89 displays the total number of calls the server received from the specific client and the average server response time in milliseconds, including the XCF transmit and CPU time of the reply. `XCF Req` is the count of XCF messages that had to be sent to other systems (most likely for token revokes) to process the client request. `Qwait` counts the number of times a wait was done for an available zFS thread to process the client request.

Note: The output is displayed only when the zFS svi component on this system has received a message from another system.

```
          SVI Calls from System NP1
          ----------------------------
SVI Call                   Count      Qwait    XCF Req.  Avg. Time
--------------------     ----------  --------  --------  ----------
GetToken                    663624         2    180593       4.246
GetMultTokens                    0         0         0       0.000
ReturnTokens                   814         0         0       8.139
ReturnFileTokens                 0         0         0       0.000
FetchData                   132962         0     13222       1.016
StoreData                  1401717         9         0       0.229
Setattr                     228600         0         0       0.527
FetchDir                         5         0         0       0.188
Lookup                       93113         1      1934       2.875
GetTokensDirSearch               0         0         0       0.000
Create                           1         0         1       5.056
Remove                           1         0         1       9.040
Rename                           0         0         0       0.000
Link                             0         0         0       0.000
ReadLink                         0         0         0       0.000
SetACL                           0         0         0       0.000
Statfs                          14         0         0       0.448
TSR                              0         0         0       0.000
FilesysSyncTable                 0         0         0       0.000
FileSyncMeta                     3         0         0       0.097
BitmapReserve                    0         0         0       0.000
BitmapUnreserve                  0         0         0       0.000
BitmapReclaim                    0         0         0       0.000
FileUpdateIB                     0         0         0       0.000
FileCreateIB                     0         0         0       0.000
FwdReaddir                       0         0         0       0.000
LkupInvalidate                   0         0         0       0.000
FileDebug                        0         0         0       0.000
FetchPage                        0         0         0       0.000
ServerIO                         0         0         0       0.000
BulkFetchStatus               5563         0         0       4.404
Convert                          0         0         0       0.000
ConvertFID                       0         0         0       0.000
------------              ----------  --------  --------  ----------
*TOTALS*                   2520851        12    195751       1.557
```

Figure 37. Sample SVI report

VM

| The VM report shows the statistics relating to the performance of the zFS user file
| caching system. The size of this cache is controlled by the IOEFSPRM
| `user_cache_size` configuration option or the **`zfsadm config`** command.

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

At any given time, a file need not (and for large files often might not) have all of its segments in the cache. Furthermore, any segment need not (and often might not) have all of its pages in the cache. Reuse of pages and segments is done in a least-recently used (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 64 K).

Following is a sample VM report.

```
IOEZ00438I Starting Query Command VM.
                   User File (VM) Caching System Statistics
                   ----------------------------------------

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

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

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

Page Management (Segment Size = 64K) ) (Page Size = 4K)
----------------------------------------------------
Total Pages       12800     Free             12800
Segments          2373
Steal Invocations      24146     Waits for Reclaim       62038

Number of dataspaces used:     1  Pages per dataspace:       12800

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

Figure 38. Sample VM report

Table 8 on page 91 describes the fields of the User File (VM) Caching System Statistics report.

Table 8. User File (VM) Caching System Statistics report fields

Field name	Contents
External Requests:	Describes the requests that are 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:	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** 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** Count of how often a write to a file needed to perform a read from disk. If a write only updates a portion of a page of a file on disk and that page is not in memory, then the page must be read in (the zFS I/O driver can only perform I/O in whole pages) before the new data is written to the in-memory page. **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** Total number of file system reads made for any reason.

Table 8. User File (VM) Caching System Statistics report fields (continued)

Field name	**Contents**
File System Writes:	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** Count of how often the cache wrote out dirty segments for a file. Segments are written as soon as every page becomes dirty (segments are said to be dirty if they contain live blocks). When a file is closed all of its dirty segments are scheduled asynchronously and segments are also written asynchronously during file system syncs through the zFS sync daemon (which by default runs every 30 seconds). **Sync Waits** Count of how often a fsync request that is needed to wait on pending I/O for dirty segments. **Error Writes and Error Waits** Count of the error handling paths and should almost always be 0 unless a disk hardware error occurs. Whenever an unexpected error occurs for a file, all of its dirty segments are written and synced to disk. (A file system that is running out of space is not an error condition that causes the cache to sync a file, the cache reserves storage for files as they are written which ensures no unexpected out of space conditions arise). **Scheduled Deletes** Count of times a pending I/O was canceled because a file was being deleted. In this case, the data is not appropriate to be on disk (because the file is 0 link count). Therefore, canceling the I/O is done to avoid an I/O wait. This is a performance optimization for file remove. **Page Reclaim Writes** Count of times that a segment had to be written to DASD to reclaim space in the cache. **Page Reclaim Waits** Count of times that the reclaim function waited on pending I/O to reclaim segment pages. **Write Waits** Count of times a write occurred to a page that was already pending I/O. In this case, the I/O must be waited upon before the page is updated with the new data.

Table 8. User File (VM) Caching System Statistics report fields (continued)

Field name	Contents
Page Management:	Shows how storage in the user file cache is used. It is generally desirable to minimize the number of steal invocations (reclaims). To minimize the number of steal invocations, increase the size of the cache. Performance is increased as more data spaces are used. **Total pages** The number of 4 K pages in the cache. That is, (`user_cache_size` / 4K). **Free** The number of available 4 K pages in the cache. **Segments** The number of 64 K sections that was referenced in a file. The number of segments starts out as half of `vnode_cache_size` and is allocated as needed, similar to vnodes. **Steal Invocations** The number of times 4 K pages were reclaimed from the cache. **Waits for Reclaim** The number of times a task waited for space to be reclaimed from the cache. **Number of data spaces** The number of data spaces that are used to hold the 4 K pages in the cache. The pages are spread evenly across the data spaces to allow for better performance of the cache. The number of data spaces that are used is approximately one per 16384 4 K pages, up to a maximum of 32. **Pages per dataspace** The number of 4 K pages that is assigned to each data space.

Debugging aids for zFS

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

Overview of trace options for zFS

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

Steps for tracing on zFS

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

1. Allocate the trace output data set as a PDSE, RECFM=VB, LRECL=133 with a primary allocation of at least 50 cylinders and a secondary allocation of 30 cylinders.
2. Define the zFS trace output data set to zFS by either using the IOEFSPRM trace_dsn option, or dynamically by using the **`zfsadm config -trace_dsn`** command.

   If you use the IOEFSPRM option, zFS must be stopped and then restarted to pick up the change, unless you also dynamically activate the trace output data set with the **`zfsadm config -trace_dsn`** command.
3. When you are ready to re-create the problem, reset the zFS trace table using the MODIFY ZFS,TRACE,RESET command.

4. Re-create the problem.
5. Enter the MODIFY ZFS,TRACE,PRINT command. This formats and prints the trace table to the PDSE defined on the `trace_dsn` option.
6. Capture the ZFSKNT*nn* 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 51.
2. Allocate the data set as a PDSE, RECFM=VB, LRECL=133 with a primary allocation of at least 50 cylinders and a secondary allocation of 30 cylinders. Each trace output is created as a new member with a name of ZFSKNT*nn*, where *nn* starts at 01 and increments for each trace output until zFS is restarted. After restart, when the next trace output is sent to the trace output data set, ZFSKNT01 is overlaid. You should not be accessing the trace output data set while a trace is being sent to the trace output data set. The space that is used by a particular trace depends on how large the `trace_table_size` is and how recently the trace was reset. For example, a 32-MB `trace_table_size` can generate a trace output member of 100 cylinders of 3390. It is important that the trace output data set be large enough to hold the trace output. If it runs out of room while sending the trace to the trace output data set, the complete trace will not be captured.

   | **Note:** You can have a `trace_table_size` up to 65535 MB, but to print the trace to a PDSE you must limit its size to 750 MB.

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

- Add events to trace by specifying the `ioedebug` statements in a data set that is read when zFS is started (or restarted). The data set name is specified in the IOEFSPRM `debug_settings_dsn` option. It is a PDS member with an LRECL of at least 80. IBM specifies the exact statements needed in the data set.
- Dynamically add the events to trace by entering the MODIFY ZFS,IOEDEBUG command. IBM specifies the exact statements needed.
- If you were not able to capture the trace, but you have a zFS dump, IBM service can obtain the trace from the dump. To obtain a dump, you can issue a MODIFY ZFS command. See "Understanding zFS dumps" on page 96 for additional information.

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

Understanding the salvager utility

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

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

- An internal error has occurred during zFS processing for the aggregate.

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

  In this case, zFS issues messages IOEZ00001E or IOEZ00550E and the message IOEZ00422E. zFS detected the I/O error and disabled the aggregate. This is most likely a hardware problem. Follow your local procedures for analyzing I/O problems to determine if you want to run the salvage utility. If you run the utility, you must first unmount the aggregate.
- A zFS problem occurs during a mount of a zFS aggregate.

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

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

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

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

- If a fundamental aggregate structure is corrupted, the salvager will not be able to recover the aggregate.
- If the aggregate is large or has many objects, the salvager might not be able to complete successfully. Even when the salvager is successful, an aggregate with many objects will take a long time to examine and attempt to repair. It might take less time to restore a backup copy of the aggregate than to salvage it.

The salvager is designed to make all repairs in one pass, but due to the nature of the program's inputs (a corrupted, possibly vastly corrupted file system) IBM recommends a second running of the salvage program to verify that the aggregate is truly repaired. If verifying the aggregate shows that it is not repaired, then you should try running the salvager again to repair the aggregate. If this does not repair the aggregate, you can create a copy of the aggregate and run the salvager more times to try and repair it. If the salvager cannot repair the aggregate after several repair attempts, the copy of the aggregate and salvager job logs will allow IBM service to determine why.

It is important to maintain backups of zFS aggregates to restore in case of a corrupted aggregate. It is also very important to maintain a regular backup regimen (for example, daily, weekly, monthly) so that if a recent backup is corrupted, you can use an older backup. However, if a quiesce is not done before backup, corruption of the file system can result. See Chapter 6, “Copying or performing a backup of a zFS,” on page 61 for recommendations for backing up zFS aggregates.

Understanding zFS dumps

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

```
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. 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 105). If zFS does stop, zFS attempts to perform an internal restart after the terminating exception occurs. If the internal restart is unsuccessful, zFS attempts a stop and restart sequence. If the restart is successful, you might need to remount any zFS file systems. You might need to remount zFS file systems. The SETOMVS command can be used to remount file systems that were mounted from a BPXPRMxx parmlib member statement.

If a failure of a zFS operation occurs (other than a user error), but zFS does not dump, you should get a trace of the failure, if possible. Perform the steps outlined in “Steps for tracing on zFS” on page 93.

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 that the dump is complete. Partial dumps are of little use.

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

Determining the XCF protocol interface level

Beginning with z/OS V2R2, zFS uses the enhanced log and enhanced status APIs XCF communication protocols and runs with `sysplex=filesys`. This change requires toleration support on V1R13 and V2R1. For more information, see *z/OS Migration*.

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

4	One of the following: • The z/OS V2R2 level (with XCF protocol enhance log and enhanced status APIs). • The z/OS V2R1 level (with XCF protocol extended directory). • The z/OS V1R13 level (with XCF enhanced connect protocol).
3	The z/OS V1R12 level.
2	The V1R12-compatible level that is used by z/OS V1R9 and z/OS V1R10.

Saving initialization messages in a data set

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

Determining service levels

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

```
IOEZ00559I zFS kernel: Initializing z/OS    zFS
Version 02.02.00 Service Level 0000000 - HZFS420.
Created on Fri Mar 20 09:26:25 EDT 2015.
Address space asid x44
```

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

```
IOEZ00639I zFS kernel: z/OS    zFS
Version 02.02.00 Service Level 0000000 - HZFS420.
Created on Fri Mar 20 09:26:25 EDT 2015.
sysplex(filesys,rwshare) interface(4)
```

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

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

```
| IOEZ00020I zfsadm: z/OS    zFS
| Version 02.02.00 Service Level 0000000 - HZFS420.
| Created on Fri Mar 20 09:26:25 EDT 2015.
```

Understanding namespace validation and correction

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

zFS namespace validation is invoked in one of four ways:

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

zFS namespace validation compares the information that is stored in each zFS member. If zFS validation detects an inconsistency, one or more messages can occur (for example, IOEZ00612I) and zFS attempts to correct the inconsistency, using one of the following actions:

- Updating the inconsistent information
- Automatically remounting a file system
- Internally restarting zFS on one or more members.

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

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

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

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 removes zFS message from the console. If the zFS message is removed, it means that the delay has cleared and was just a slowdown because of a stressful workload or some other issue. In this case, you can discard any zFS dumps that occur because of this delay.

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

IOEZ00*xxx*I zFS messages are issued by the zFS hang detector and generally remain on the console until the situation is resolved. Resolution occurs when:

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

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

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

- If the other system never received the XCF message, zFS issues message IOEZ00591I.
- If 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.
- If the other system received the XCF message but the progress is very slow or long running, zFS issues message IOEZ00661I.
- If the other system processed the XCF message and sent a response back, but zFS did not receive the response, zFS issues message IOEZ00592I.

In these cases, zFS does not issue a system dump. 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 timeouts on each XCF message. Wait to see whether a request timing out resolves the hang. If a request times out, the request will fail.

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

Steps for diagnosing and resolving a zFS hang

About this task

Perform the following steps when a hang condition is suspected.

Procedure

1. Continually monitor for the following messages:

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

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

   To start investigating, if in a sysplex file sharing environment check for message IOEZ00547I (hanging XCF request), which can indicate an XCF issue. If you see this message:

   a. Check the status of XCF on each system in the sysplex.
   b. Check for any outstanding message that might need a response to determine whether a system is leaving the sysplex or not (for example, IXC402D). The wait for a response to the message might appear to be a zFS hang.

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

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

   IOEZ00605I
   The delay is either in zFS or in a system service that zFS did not specifically identify in message IOEZ00604I. zFS cannot determine whether there is a hang, a slowdown, or some other system problem. To take action, look for other symptoms. For example, if you see messages

about components that are using a significant amount of auxiliary storage, resolve the auxiliary storage shortage. If the message persists, continue to the next step.

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

   The type and amount of information that is displayed as a result of this command is for internal use and can vary between releases or service levels. For an example, see Figure 39 on page 102.
3. Enter the DISPLAY A,ZFS command to determine the zFS ASID.
4. Enter MODIFY ZFS,QUERY,THREADS at one to two-minute intervals for six minutes.
5. Check the output for any user tasks (tasks that do not show the zFS ASID) that are repeatedly in the same state during the time you requested MODIFY ZFS,QUERY,THREADS. If there is a hang, the task that is hanging persists unchanged over the course of this time span. If the information is different each time, there is no hang.
6. If message IOEZ00581E is highlighted in white on the console, there are or recently were quiesced zFS aggregates. Verify that no zFS aggregates are in the QUIESCED state by checking their status using the **zfsadm lsaggr**, **zfsadm aggrinfo -long**, or **zfsadm fsinfo** command. For example, quiesced aggregates are displayed as follows:

```
DCESVPI:/home/susvpi/> zfsadm lsaggr
IOEZ00106I A total of 1 aggregates are attached
SUSVPI.HIGHRISK.TEST                        DCESVPI   R/W QUIESCE
DCESVPI:/home/susvpi/> zfsadm aggrinfo
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/>
```

   or

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

   This example shows how to determine which aggregates are quiesced with the owner information.

```
> ./zfsadm fsinfo -select Q
PLEX.DCEIMGNJ.FS4                        DCEIMGNJ RW,RS,Q
PLEX.DCEIMGNK.FS6                        DCEIMGNK RW,RS,Q

Legend: RW=Read-write,Q=Quiesced,RS=Mounted RWSHARE
```

   If the hang condition prevents you from issuing shell commands, you can also issue the MODIFY ZFS,QUERY,FILE,ALL command to determine whether any file systems are quiesced. As Figure 30 on page 74 shows, a quiesced file system is identified by a “Q” in the `flg` column.

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

Message IOEZ00581E appears on the zFS owning systems that contain at least one zFS aggregate that is quiesced. There is a delay between the time that the aggregate is quiesced and the time that the message appears. Typically, this time delay is about 30 seconds. You can control this time delay by using the IOEFSPRM QUIESCE_MESSAGE_DELAY option. This option allows you to specify that the delay should be longer than 30 seconds before the IOEZ00581E message is first displayed. When there are no quiesced zFS aggregates on the system, this message is removed from the console.

There is also a delay between the time that the last aggregate is unquiesced and the time that the message is removed from the console. This message is handled by a thread that wakes up every 30 seconds and checks for any quiesced aggregates that are owned by this system. It is possible for an aggregate to be quiesced and unquiesced in the 30-second sleep window of the thread and not produce a quiesce message. This message remains if one aggregate is unquiesced and another is quiesced within the 30-second sleep window.

7. Check whether any user tasks are hung, focusing on the tasks that are identified by message IOEZ00524I or message IOEZ00660I. User tasks do not have the same address space identifier (ASID) as the zFS address space. One or more threads consistently at the same location might indicate a hang (for example, Recov, TCB, ASID Stack, Routine, State). The threads in the zFS address space with the zFS ASID (for example, xcf_server) are typically waiting for work. It is typical for the routine these threads are waiting in to have the same name as the entry routine. For an example, see Figure 39.

   If successive iterations of the MODIFY ZFS,QUERY,THREADS command show that the STK/Recov, TCB, ASID, Routine, and State for a thread are constant, it is probable that this thread is hung.

Figure 39. Example of how to check whether user tasks are hung

```
zFS and z/OS UNIX Tasks
            -----------------------
STK/Recov     TCB     ASID   Stack      Routine              State
----------  --------  ----  ----------  --------             --------
48338F0000  005CABE8  005A  48338F0700  ZFSRDWR              OSIWAIT
48000AF8F0
       since  Oct 14 04:15:57 2014  Current DSA: 48338F2D38
       wait code location offset=0ACA rtn=allocate_pages
  wait for resource=7BCC6330 0
       resource description=VNOPS user file cache page reclaim wait
    ReadLock held for 4823FDBF50 state=2 0
       lock description=Vnode-cache access lock
  Operation counted for OEVFS=7E7EC190  VOLP=4826660200
fs=PLEX.ZFS.SMALL1

48338E8000  005CA1D0  00B8  48338E8810  ZFSCREAT             WAITLOCK
48000B0640
       since  Oct 14 04:15:57 2014  Current DSA: 48338EB5C8
       wait code location offset=3D74 rtn=epit4_Allocate
  lock=48203E30F0 state=80000048000D6AA1 owner=(48000D6AA0 00B85CA830)
       lock description=ANODETB status area lock
    ReadLock held for 4833F0DE50 state=A 0
       lock description=Vnode-cache access lock
    ReadLock held for 4833F0DEC0 state=8 0
       lock description=Vnode lock
    ReadLock held for 482060CC20 state=7 7A94FEF0
       lock description=Vnode lock
    ReadLock held for 482606BA00 state=4 0
       lock description=Anode fileset handle lock
    ReadLock held for 48203E30E0 state=4 0
```

```
      lock description=ANODETB main update lock
   Resource 4833F0DE40 1A held
      resource description=STKC held token by local user task
   Resource 4826661800 17 held
      resource description=ANODE maximum transactions started for a
   Resource 4830D68580 2F held
      resource description=Transaction in progress
  Operation counted for OEVFS=7AB8DA20  VOLP=4826661A00
fs=ZFSAGGR.BIGZFS.DHH.FS1.EXTATTR

48338E0000  005C12F8  0084  48338E0700  ZFSRDWR              WAITLOCK
48000B1390
       since  Oct 14 04:15:57 2014  Current DSA: 48338E23C8
       wait code location offset=4940 rtn=stkc_getTokenLocked
  lock=4823F8CFD0 state=5 owner=(2 read holders)
       lock description=Vnode-cache access lock
  Operation counted for OEVFS=7AB8D1E0  VOLP=4826663200
fs=ZFSAGGR.BIGZFS.DHH.FS6.EXTATTR

48338D8000  005CAD80  0079  48338D8700  ZFSRDWR              OSIWAIT
48000B20E0
       since  Oct 14 04:15:57 2014  Current DSA: 48338DAE38
       wait code location offset=0ACA rtn=allocate_pages
  wait for resource=7BCC6330 0
       resource description=VNOPS user file cache page reclaim wait
    ReadLock held for 4823F49F10 state=A 0
       lock description=Vnode-cache access lock
  Operation counted for OEVFS=7AB8D1E0  VOLP=4826663200
fs=ZFSAGGR.BIGZFS.DHH.FS6.EXTATTR

48338D0000  005CAA50  00B7  48338D0810  ZFSCREAT             RUNNING
48000B2E30
       since  Oct 14 04:15:57 2014
    ReadLock held for 7E5C2670 state=2 0
       lock description=Cache Services hashtable resize lock
    Resource 4823FF4820 1A held
       resource description=STKC held token by local user task
    Resource 4826661E00 17 held
       resource description=ANODE maximum transactions started for a
    Resource 4831569A80 2F held
       resource description=Transaction in progress
  Operation counted for OEVFS=7AB8D810  VOLP=4826662000
fs=ZFSAGGR.BIGZFS.DHH.FS2.EXTATTR

48338C8000  005CABE8  00A6  48338C8700  ZFSRDWR              OSIWAIT
48000B3B80
       since  Oct 14 04:15:57 2014  Current DSA: 48338CAD38
       wait code location offset=0ACA rtn=allocate_pages
  wait for resource=7BCC6330 0
       resource description=VNOPS user file cache page reclaim wait
    ReadLock held for 4835B3ABD0 state=6 0
       lock description=Vnode-cache access lock
  Operation counted for OEVFS=7E7EC190  VOLP=4826660200
fs=PLEX.ZFS.SMALL1

  7F37B000  005D5528  0044    7F37C000  openclose_task       RUNNING
       since  Oct 14 03:43:35 2014

  7F3B4000  005F81D0  0044    7F3B5000  CNMAIN               WAITING
       since  Oct 14 02:58:01 2014

  7BC45000  005C19C0  0044    7BC46000  comm_daemon          RUNNING
4800004290
       since  Oct 14 04:15:57 2014
```

8. IBM Support must have dumps of zFS, OMVS and the OMVS data spaces and also possibly the user address space identified on any preceding IOEZ00605

for problem resolution. Obtain and save SYSLOG and dumps of zFS, OMVS and the OMVS data spaces , and the user ASID using `JOBNAME=(OMVS,ZFS,user_jobname),DSPNAME=('OMVS'.*)` in your reply to the DUMP command. If you are running in a sysplex and zFS is running on other systems in the sysplex, dump all the systems in the sysplex where zFS is running, dumping zFS, OMVS and OMVS data spaces. The following is an example of the DUMP command:

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

Do not specify the job name ZFS if zFS is running inside the OMVS address space.

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

9. If you know which user task is hung (for example, returned in IOEZ00524I or determined to be hung after review of the output from repeated MODIFY ZFS,QUERY,THREADS,OLDEST commands), consider entering the CANCEL or STOP command to clear that task from the system.
10. Finally, if the previous steps do not clear the hang, issue the MODIFY ZFS,ABORT command to initiate a zFS internal restart.

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

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

Identifying storage shortages in zFS

When zFS can no longer obtain sufficient storage to complete a request, it issues message IOEZ00188A, possibly creates a dump, and restarts. If you see message IOEZ00188A before zFS initialization is complete (before message IOEZ00055I), either increase the REGION size in the ZFS PROC or decrease some cache sizes in the 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. See “STOR” on page 85 for more information about how to determine the amount of storage being used in the zFS address space.

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

Tips:

- Changing the size of a cache can cause delays. Try to change the size during low activity periods.
- In general, if you see a return code of 132 (ENOMEM), zFS is short on storage; take steps to reduce zFS storage usage. When storage shortages become critical, you can also see 157 (EMVSERR) and mounts might begin to fail.
- Started subtasks, such as the zFS colony address space, fall under SUBSYS STC. These address spaces might be subject to IEFUSI limitations if IEFUSI exits are allowed for SUBSYS STC. IBM strongly recommends that you always set REGION=0M and MEMLIMIT=NOLIMIT for the zFS colony address space.

Diagnosing disabled aggregates

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

```
IOEZ00422E Aggregate PLEX.JMS.AGGR001.LDS0001 disabled
```

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

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

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

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

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

An aggregate that was disabled might be corrupted, even if it was disabled and remounted. To be sure that the aggregate is internally consistent, run the **ioefsutl salvage** batch utility against the aggregate that was disabled, to repair any corruption, and prevent loss of data. See "ioefsutl" on page 136 for more information.

Handling disabled aggregates

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

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

The following information applies when some systems are on z/OS V1R13 or later.
| If a compatibility mode aggregate becomes disabled, zFS attempts to automatically
| re-enable the disabled aggregate. It either requests that another system in the
| shared file system environment take over the aggregate (if it is sysplex-aware) or it attempts an internal remount samemode. This action should recover the aggregate and it will no longer be disabled.

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

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

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

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

Chapter 9. Overview of the zFS audit identifier

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

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

Figure 40 shows the format of the unique zFS auditid, the standard zFS auditid, and the HFS auditid.

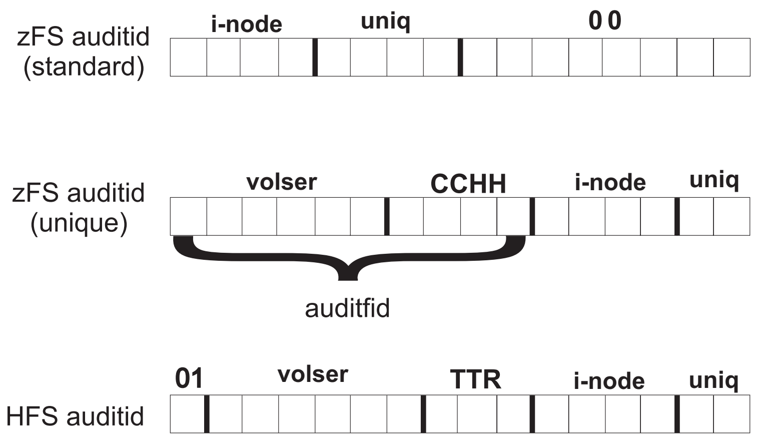

Figure 40. zFS auditid examples

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

The auditfid in the zFS aggregate controls the type of auditid zFS uses: unique auditid or less unique auditid (auditfid of binary zeros). Typically, a zFS aggregate contains a zero auditfid, but you can take steps to store a unique zFS auditfid, which subsequently causes zFS to generate a unique format auditid for each file or directory in the aggregate.

There are three ways to control the zFS auditfid that is stored in the aggregate, which thereby controls the format of the zFS auditid for files and directories that are contained in the aggregate:

- When formatting an aggregate, you get a unique auditfid by default (that is, if you do not specify `-nonewauditfid`). This is true for the IOEAGFMT batch utility and the **`zfsadm format`** command. If you specify `-nonewauditfid`, the aggregate has the standard auditfid (binary zeros). The IOEFSUTL format always provides a unique auditfid.
- You can optionally specify a zFS configuration option (convert_auditfid=on) in the IOEFSPRM file to control whether the aggregate's auditfid is converted from a standard format auditfid to a unique auditfid when a zFS file system is mounted. If you specify on, zFS converts the standard auditfid to the unique auditfid on the read/write mount (attach) of the aggregate. You can also specify the convert_auditfid configuration option using the **`zfsadm config -convert_auditfid`** option and query using the **`zfsadm configquery -convert_auditfid`** option. The default for convert_auditfid is ON.
- You can explicitly set an aggregate's auditfid to a unique auditfid using the **`zfsadm setauditfid`** command.

Enabling the zFS auditid

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

Scenarios:

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

   Result: Any existing aggregates are converted to the unique auditfid the next time they are mounted (attached). Newly formatted aggregates using IOEAGFMT, or **`zfsadm format`** get unique auditfids by default. IOEFSUTL format always creates unique auditfids.
2. You want your new aggregates to have the unique auditfid and your existing aggregates to remain with the standard auditfid:
   a. Specify `convert_auditfid=off` in your IOEPRMxx configuration file.
   b. Specify (or default to) `-newauditfid` when you format new aggregates using IOEAGFMT or **`zfsadm format`**. Use IOEFSUTL to format new aggregates.

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

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

Tip: New aggregates formatted with ISHELL, automount allocany, allocuser, or the BPXWH2Z utility will not have unique auditfids after they are formatted. However, they will be converted to unique auditfids by default the first time they are mounted unless you specify `convert_auditfid=off` in your IOEPRMxx configuration file or specify **`zfsadm config -convert_auditfid off`**.

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

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

Use the following algorithm to help distinguish between the unique auditfid, the standard zFS auditfid, and HFS auditid (which 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 contains reference information for zFS, and includes the following topics:

Chapter 10. z/OS system commands

Several z/OS system commands are available.

- MODIFY ZFS PROCESS queries internal counters and values. Use it to initiate or gather debugging information.
- SETOMVS RESET starts the zFS Physical File System (PFS) if it has not been started at IPL, or if the PFS was stopped and the BPXF032D message was responded to with a reply of `i`.

Run these commands from the console or from System Display and Search Facility (SDSF).

MODIFY ZFS PROCESS

Purpose

The MODIFY ZFS PROCESS command 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. To use this command, the zFS PFS must be running.

| Prior to z/OS V2R2, zFS always ran as a colony address space. The syntax of that command was **modify zfs,<cmd>**.

| Beginning in z/OS V2R2, zFS can be run as a colony address space or in the OMVS address space. In both cases, the syntax of the modify command can be **modify omvs,pfs=zfs,<cmd>**. This form of the modify command should also be used if you have any zFS modify commands issued through an automated process or system automation.

| When zFS modify commands in this documentation are mentioned, they are shown in the historical **modify zfs,<cmd>** form, as they always have been, rather than always mentioning both forms.

Format

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

```
modify procname,query,{level | settings | threads[,{allwait | oldest}] | status|
[{kn | vm | lfs | lock | storage | file | stkm | ctkc | svi  | iobydasd | dataset | all}]

modifyprocname,reset,{kn | vm | lfs | lock | storage | file | stkm | ctkc | svi | iobydasd |
   dataset | all}

modify procname,trace,{reset | print}

modify procname,abort

modify procname,dump

modify procname,hangbreak

modify procname,unquiesce,aggregate_name

modify procname,nsvalidate[,print]

modify procname,fsinfo[,{aggrname | all} [,{full | basic | owner | reset}
[,{select=criteria | exceptions}] [,sort=sort_name]]]
```

Parameters

procname
: The name of the zFS PFS PROC. The default *procname* is ZFS.

  If zFS is running in the OMVS address space (the address space that is used by z/OS UNIX), *procname* must direct the command to zFS through OMVS. For example:

  ```
  modify omvs,pfs=zfs,command
  ```

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

  abort Causes zFS to dump and then perform an internal restart. The internal trace table is also printed to the data set specified in the IOEFSPRM file `trace_dsn` entry.

dump Causes the zFS PFS to dump and to print the internal trace table to the data set specified in the IOEFSPRM file `trace_dsn` entry.

fsinfo Displays detailed information about a zFS file system, which is also known as a *zFS aggregate*.

aggrname
Specifies the name of the aggregate that the detailed zFS information is for. The aggregate name is not case-sensitive and is translated to uppercase. To specify multiple aggregates with similar names, use an asterisk (*) at the beginning, at the end, or both at the beginning and the end of *aggrname* as a wildcard. If *aggrname* is specified with wildcards, the default display is **basic**. Otherwise, the default display is **owner**. For more information, see "Usage notes for displaying file system information" on page 121 and "Examples of displaying file system information" on page 122.

all Displays information for all aggregates in the sysplex. It is the default when *aggrname* is not specified. The default information display will be as if **basic** were specified.

basic Displays a line of basic file system information for each specified file system. This option is the default in the following situations:

- The **all** option is specified but **full**, **owner**, and **reset** are not specified.
- If *aggrname* and **all** are not specified.
- *aggrname* is specified with wildcards.

For more information about what is displayed when the **basic** option is used, see Table 13 on page 194.

exceptions
Displays information about any specified aggregate that is quiesced, disabled, had grow failures, is low on space, failed to convert a directory to version5, or is damaged. Any specified aggregate is also displayed if it has had XCF communication failures or an error because it ran out of space or when doing I/O. This option cannot be specified with **reset**, **select**, and *aggrname* with no wildcard.

full Displays information that is maintained by the system owning each specified file system. It also displays information that is locally maintained by each system in the sysplex that has each specified file system locally mounted.

Tip: If a large number of file systems are to be displayed, a large amount of output will be displayed. For that case, consider using either the **basic** output option or the **`zfsadm fsinfo`** command so that the output can be redirected to a file.

owner Displays only information that is maintained by the system owning each file system specified. This option is the default when *aggrname* with no wildcards is specified. For more information about what is displayed when the **owner** option is used, see Table 13 on page 194 and Table 14 on page 194.

Tip: If a large number of file systems are to be displayed, a large amount of output will be displayed. For that case, consider using either the **basic** output option or the `zfsadm fsinfo` command so that the output can be redirected to a file.

reset Resets zFS statistics that relate to each specified file system. `reset` cannot be specified with **basic**, **full**, **owner**, **exceptions**, **select**, or **sort**.

select=*criteria*
Displays each specified file system that matches the criteria.

This option cannot be specified with **exceptions**, **reset**, and *aggrname* with no wildcard.

To use this option, specify a selection criteria from Table 12 on page 193. Multiple criteria are separated by spaces.

sort=*sort_option*
Sorts the displayed information using the value of *sort_option*. The default is to sort by Name. This option cannot be specified with **reset**. For a list of the sorting options, see Table 15 on page 196.

hangbreak
Causes a zFS internal restart; this produces the same result as issuing a `modify zfs,abort` command.

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

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

query Displays zFS counters or values.

level Displays the zFS level for the zFS physical file system kernel. When running in a shared file system environment, `level` also displays the zFS sysplex level and the zFS XCF communication interface level (1, 2, 3 or 4). The zFS sysplex level is controlled by the IOEFSPRM sysplex configuration option. When the sysplex level is *filesys*, the default mount PARM (NORWSHARE or RWSHARE) is also displayed. (As of z/OS V1R13, zFS always runs with `sysplex=filesys`.) For an example and more information, see "Determining service levels" on page 97.

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

status
: Displays zFS internal restart information.

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

<*report*>
: One of the following report options. These parameters all produce reports; for details about these reports, see "Monitoring zFS performance" on page 71.

  all Displays all the zFS counters.

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

  dataset Displays zFS statistics about file systems.

  file Displays the requests per zFS file system and aggregate.

  iobydasd Displays the DASD that is attached by volume.

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

  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.

  log Displays the log statistics.

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

  storage Displays the zFS storage values.

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

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

reset
: Resets zFS counters and consists of the following options:

  all Resets all the zFS counters to zero.

  ctkc Resets the client call statistics.

dataset
Reset the zFS statistics about file systems.

file Resets the requests for zFS file system and aggregate.

iobydasd
Resets the count of the DASD that is attached by volume.

kn Resets the calls that were made to zFS from z/OS UNIX.

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.

log Resets the log statistics.

stkm Resets the server token manager (STKM) statistics.

storage
Resets the zFS storage counters.

svi Resets the received calls from other systems statistics.

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

| No other options are allowed after **`reset`**.

trace Resets or prints the internal zFS trace table.

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 preallocated as a PDSE with RECFM VB and LRECL 133. It must be large enough to hold the formatted trace table. See Chapter 8, "Performance and debugging," on page 67 for more information about the trace output data set.

reset Resets the internal (wrap around) trace table to empty.

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.

Usage notes for MODIFY ZFS PROCESS

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

Usage notes for displaying file system information

Use the MODIFY FSINFO command to display detailed information about zFS file systems, which are also known as *zFS aggregates*. Normally, file systems must be attached before this command can be used to display their information. However, when specifying a specific aggregate name (with no wildcards), the file system does not need to be attached. You can use several methods to specify aggregates, based on their names, as follows:

- *aggrname* with an exact aggregate name. The aggregate can either be mounted or not mounted.
- *aggrname* using a wildcard (*) at the beginning of the name value to select aggregates with a common suffix.
- *aggrname* using a wildcard (*) at the end of the name value to select aggregates with a common prefix.
- *aggrname* using a wildcard (*) at the beginning and the end of the name value to select aggregates with both a common prefix and a common suffix.
- **all** can be specified or defaulted to mean all file systems that are currently mounted in the sysplex.

The MODIFY FSINFO command options are positional. Each option must be separated by a comma. Only the options at the end of the line can be omitted. If options are omitted, the default values are used instead. Examples of supported syntax are as follows:

```
F ZFS,FSINFO
F ZFS,FSINFO,ALL
F ZFS,FSINFO,ALL,BASIC,SELECT=RW Q
F ZFS,FSINFO,ALL,BASIC,SELECT=RW Q,SORT=REQUESTS
```

The **owner** option displays all available information for each specified file system from the zFS-owning system. The information is obtained via XCF communication with the owning system if the owning system is not the local system. It also displays the statistics that are shown in Table 14 on page 194.

The **full** option displays statistics for each specified file system from the zFS owning system and from each system in the sysplex that has it locally mounted. This will be obtained via XCF communication with each system in the sysplex. The statistics are described in Table 16 on page 196.

Aggregates can also be selected using the **exceptions** option. This option can be useful for identifying file systems which have encountered unexpected conditions, and might need attention. Unexpected conditions include I/O errors, XCF communication failures or being low on space. An aggregate can also be damaged, quiesced, or disabled.

Aggregates can also be selected by use of the **select** option. To use this option, specify a criteria from the list in Table 12 on page 193. You can specify more than one criteria by using a space to separate them.

The displayed information has the file system status as part of the output. The status field contains abbreviated values. For quick reference, these values are defined in a Legend string at the end of the output. The full definitions of these abbreviations are listed in Table 13 on page 194.

All times are in milliseconds. To display large numbers, use the following suffixes:

Letter Unit of number

b	The number should be multiplied by 1,000,000,000.
G	The number should be multiplied by 1,073,741,824.
t	The number should be multiplied by 1000.
T	The number should be multiplied by 1,099,511,627,776.
tr	The number should be multiplied by 1,000,000,000,000.
m	The number should be multiplied by 1,000,000.
K	The number should be multiplied by 1024.
M	The number should be multiplied by 1,048,576.

Privilege required

This command is a z/OS system command.

Examples for MODIFY ZFS PROCESS

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:

The following example causes the zFS PFS to execute an internal restart:

```
modify zfs,abort
```

The following example queries all the zFS counters when zFS is running inside the OMVS address space:

```
modify omvs,pfs=zfs,query,all
```

Examples of displaying file system information

1. To display basic file system information for zFS aggregate PLEX.DCEIMGNK.FSINFO:
   ```
   modify zfs,fsinfo,aggr,PLEX.DCEIMGNK.FSINFO,basic
   ```
2. To display file system owner status using a wildcard:
   ```
   modify zfs,fsinfo,aggr,PLEX.DCEIMGNK.*,owner
   ```
3. To display full file system status for all zFS aggregates that are quiesced, damaged or disabled:
   ```
   modify zfs,fsinfo,all,full,select=Q DA DI
   ```
4. To display basic file system status for all zFS aggregates that are quiesced, damaged, or disabled and also to sort aggregate names by response time:
   ```
   modify zfs,fsinfo,all,basic,select=Q DA DI,sort=response
   ```

Related information

Files:
- IOEFSPRM
- **`zfsadm fsinfo`**

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

SETOMVS RESET

Purpose

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

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 later, the SETOMVS command also processes zFS FILESYSTYPE statements. For more information, see SETOMVS command in *z/OS MVS System Commands*.

Chapter 11. zFS commands

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

In addition to displaying z/OS UNIX reason codes, the z/OS UNIX shell command, **bpxmtext**, also displays the text and action of zFS reason codes (EF*xxnnnn*) 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 EF00*nnnn* 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).

Restriction: The **bpxmtext** command is not valid for zFS abend reason codes (EA*xxnnnn*).

You can use the **man** command to view the descriptions of zFS command manual pages. To use man pages, enter **man** followed by the command information you want to display. You must enter the zfsadm command suite entries as one word. Table 9 shows examples of the zFS man commands.

Table 9. zFS man command examples

zFS command	man command
ioefsutl salvage	**man ioefsutlsalvage**
ioeagfmt	**man ioeagfmt**
mount	**man zfsmount**
zfsadm aggrinfo	**man zfsadmaggrinfo**
zfsadm query	**man zfsadmquery**

For more information about the **man** command, see

- man — Display sections of the online reference manual in *z/OS UNIX System Services Command Reference*.
- Enabling the man pages in *z/OS UNIX System Services Planning*.

ioeagfmt

Purpose

| **ioeagfmt** is a batch utility that formats a VSAM linear data set to become a version
| 4 or version 5 zFS compatibility mode aggregate.

Format

```
ioeagfmt -aggregate name [-initialempty blocks] [-size blocks] [-logsize blocks]
[-overwrite] [-compat] [-owner {uid|name}][-group {gid|name}] [-perms {number}]
[-grow blocks] [{-newauditfid|-nonewauditfid}][{-version4|-version5}] [-level] [-help]
```

Options

-aggregate *name*
: Specifies the name of the data set to format. This is also the aggregate name. The aggregate name is always translated to uppercase and cannot be longer than 44 characters. The following characters can be included in the name of an aggregate:
- All uppercase and lowercase alphabetic characters (a to z, A to Z)
- All numerals (0 to 9)
- The . (period)
- The - (dash)
- The @ (at sign)
- The # (number sign)
- The $ (dollar).

-compat
: Indicates that a compatibility mode aggregate should be created. This means that in addition to formatting the VSAM linear data set as a zFS aggregate, a zFS file system is created with the same name as the aggregate and its free space is set to the size of the available blocks on the aggregate. Beginning with z/OS V2R1, only HFS compatibility mode aggregates can be created. This option is being allowed for compatibility with earlier versions and is not needed.

-group *gid* | *name*
: Specifies the group owner for the root directory of the file system. It can be specified as a z/OS group name or as a GID. The default is the GID of the issuer of **ioeagfmt**. If only `-owner` *name* is specified, the group is that owner's default group. If only `-owner` *uid* is specified, the group is the issuer's group.

-grow *blocks*
: Specifies the number of 8-KB blocks that zFS will use as the increment for extension when the `-size` option specifies a size greater than the primary allocation.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

-initialempty *blocks*
: This option is being allowed for compatibility with earlier versions and is ignored. One 8-KB block at the beginning of the aggregate is reserved for IBM use.

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

-logsize *blocks*
Specifies the size in 8-KB blocks of the log. The valid range is from 13 to 16384 blocks (128 megabytes). The default is 1% of the aggregate size. This default logsize will never be smaller than 14 blocks and it will never be larger than 4096 blocks (32 megabytes). This size is normally sufficient. However, a small aggregate that is grown to be very large will still have a small log. You might want to specify a larger log if you expect the aggregate to grow very large.

-newauditfid
Specifies that the aggregate should be formatted with the zFS auditfid and stored in the aggregate. Beginning with z/OS V2R1, `-newauditfid` is the default.

-nonewauditfid
Specifies that the aggregate should not be formatted with a zFS auditfid stored in it. Before z/OS V2R1, this was the default.

-overwrite
Required if you are reformatting an existing aggregate. Use this option with caution; it destroys any existing data. This option is not usually specified.

-owner *uid* | *userid*
Specifies the owner for the root directory of the file system. It can be specified as a z/OS user ID or as a UID. The default is the UID of the issuer of **ioeagfmt**.

-perms *number*
Specifies the permissions for the root directory of the file system. The number can be specified as octal (for example, o755), as hexadecimal (for example, x1ED), or as decimal (for example, 493). The default is o755 (owner read/write-execute, group read-execute, other read-execute).

-size *blocks*
Specifies the number of 8-KB blocks that should be formatted to form the zFS aggregate. The default is the number of blocks that will fit in the primary allocation of the VSAM linear data set. If a number less than the default is specified, it is rounded up to the default. If a number greater than the default is specified, a single extend of the VSAM linear data set is attempted after the primary allocation is formatted unless the `-grow` option is specified. In that case, multiple extensions of the amount that is specified in the `-grow` option will be attempted until the `-size` is satisfied. The size can be rounded up to a control area (CA) boundary by DFSMS. It is not necessary to specify a secondary allocation size on the DEFINE of the VSAM linear data set for this extension to occur. Space must be available on the volume.

-version4
Specifies that the aggregate should be a version 1.4 aggregate. See the Usage section for the default value that is used.

-version5
Specifies that the aggregate should be a version 1.5 aggregate. See the Usage section for the default value that is used.

Usage notes for ioegfmt

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

The **ioeagfmt** utility formats an existing VSAM linear data set as a zFS aggregate. All zFS aggregates must be formatted before use.

The aggregate version will be as specified if the `-version4` or `-version5` option is used. If neither is used, then the default aggregate version will be obtained from the zFS PFS *format_aggrversion* setting. See "IOEFSPRM" on page 220 for a description of the *format_aggrversion* variable.

The size of the aggregate is as many 8-KB blocks as fits in the primary allocation of the VSAM linear data set or as specified in the `-size` option. The `-size` option can cause one additional extension to occur during formatting. To extend it further, use the **zfsadm grow** command. If `-overwrite` is specified, all existing primary and secondary allocations are formatted and the size includes all of that space. If the VSAM linear data set has a SHAREOPTIONS value of other than 3, **ioeagfmt** changes it to SHAREOPTIONS 3 during format. `-overwrite` will also cause the backup change activity flag to be set.

For a batch job, the **ioeagfmt** options are specified in the EXEC PARM as a single subparameter (a single character string enclosed in apostrophes with no commas separating the options). You cannot put the ending apostrophe in column 72. If it needs to go to the next line, use a continuation character in column 72 (continuing in column 16 with the ending apostrophe on the second line). Remember that a JCL EXEC PARM is limited to 100 characters. See the topic on the EXEC PARM in *z/OS MVS JCL Reference*.

Privilege required

If you are using an IOEFSPRM file in your ZFS PROC, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

The user must meet one of the following authorization requirements:
- Have ALTER authority to the VSAM linear data set.
- Be UID 0.
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

UPDATE authority to the VSAM linear data set is sufficient for format, but zFS will not be able to set the zFS bit in the catalog unless the issuer has ALTER authority.

If you are changing the owner or group to something other than the issuer or you are changing the permissions to other than the default, you need UID 0 or READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Examples

Figure 41 on page 129 shows an example of a job that creates a compatibility mode aggregate and file system.

```
//USERIDA  JOB ,'Compatibility Mode',
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE   EXEC   PGM=IDCAMS
//SYSPRINT DD     SYSOUT=H
//SYSUDUMP DD     SYSOUT=H
//AMSDUMP  DD     SYSOUT=H
//DASD0    DD     DISP=OLD,UNIT=3390,VOL=SER=PRV000
//SYSIN    DD     *
     DEFINE CLUSTER (NAME(OMVS.PRV.COMPAT.AGGR001) -
            VOLUMES(PRV000) -
            LINEAR CYL(25 0) SHAREOPTIONS(3))
/*
//CREATE   EXEC   PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate OMVS.PRV.COMPAT.AGGR001')
//SYSPRINT DD     SYSOUT=H
//STDOUT   DD     SYSOUT=H
//STDERR   DD     SYSOUT=H
//SYSUDUMP DD     SYSOUT=H
//CEEDUMP  DD     SYSOUT=H
//*
```

Figure 41. Sample job to create a compatibility mode aggregate and file system

Requirement: In the `PARM=('-aggregate OMVS.PRV.COMPAT.AGGR001')` statement, the `-aggregate` option must be in lowercase.

ioeagslv

Purpose

ioeagslv is a batch utility that scans an aggregate and reports inconsistencies. Aggregates can be verified, recovered (that is, the log is replayed), or salvaged (that is, the aggregate is repaired). This utility is known as the salvager.

This utility is not normally needed. If a system failure occurs, the aggregate log is replayed automatically, the next time the aggregate is attached or mounted. This action typically brings the aggregate back to a consistent state. The aggregate must not be mounted (or attached) when **ioeagslv** is run.

Format

```
ioeagslv -aggregate name [{-recoveronly | -verifyonly | -salvageonly}]
[-verbose] [-level] [-help]
```

Options

-aggregate *name*
: Specifies the name of the aggregate to be verified, recovered, or salvaged. The aggregate name is not case-sensitive. It is translated to uppercase.

-help
: Prints the online help for this command. All other valid options that are specified with this option are ignored.

-level
: Prints the level of the **ioeagslv** command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

-recoveronly
: Directs the salvager to recover the specified aggregate. The salvager replays the log of metadata changes that resides on the aggregate. See "Usage notes for ioeagslv" on page 131 for information about using and combining the command's options.

-salvageonly
: Directs the salvager to salvage the specified aggregate. The salvager attempts to repair any inconsistencies it finds on the aggregate. See "Usage notes for ioeagslv" on page 131 for information about using and combining the command's options.

-verbose
: This option is ignored.

-verifyonly
: Directs the salvager to verify the specified aggregate. The salvager examines the structure of the aggregate to determine if it contains any inconsistencies, reporting any that it finds. See "Usage notes for ioeagslv" on page 131 for information about using and combining the command's options.

Results

The salvager returns the following return codes for `-verifyonly`:

Table 10. Return codes for -verifyonly that are returned by the salvager

Code	Description
00	Success. The aggregate is correct and no repair is needed.
04	The aggregate has some inconsistencies that need repair.
08	An error occurred during verification; the report might be incomplete.
12	A severe error occurred during verification. Verify that processing was halted. The aggregate is not repairable.
16	Terminating error.
EIO	The salvager could not read or write the DASD.
EBUSY	The aggregate was mounted or attached.
EMVSERR	The salvager had an internal error. This return code is preceded by a dump for an abend 2C3 and reason code EA660701.
ENOMEM	The salvager ran out of storage.
EINVAL	The salvager arguments were incorrect.
ENOSPC	Dynamic grow failed because the salvager ran out of disk space.

For no options specified (or the `-recoveronly` and `-salvageonly` options specified) the salvager returns the following return codes:

Table 11. Return codes for -recoveronly that are returned by the salvager

Code	Description
00	Success. The aggregate is correct and no repair is needed.
04	The aggregate had some inconsistencies that were repaired.
08	An error occurred during verification; the report might be incomplete; the aggregate could not be repaired.
12	A severe error occurred during verification and the aggregate could not be repaired. Verification processing was stopped..
16	Terminating error.
EIO	The salvager could not read or write the DASD.
EBUSY	The aggregate was mounted or attached.
EMVSERR	The salvager had an internal error. This return code is preceded by a dump for an abend 2C3 and reason code EA660701.
ENOMEM	The salvager ran out of storage.
EINVAL	The salvager arguments were incorrect.

Usage notes for ioeagslv

You can run **`ioeagslv`** even if the zFS PFS is not active on the system. The **`ioeagslv`** utility invokes the salvager on the zFS aggregate that is specified with the `-aggregate` option. After a system restart, the salvager employs the zFS file system log mechanism to return consistency to a file system by running recovery on the aggregate on which the file system resides. Recovery is the replaying of the log on the aggregate; the log records all changes that are made to metadata as a result of operations such as file creation and deletion. If problems are detected in the basic structure of the aggregate, if the log mechanism is damaged, or if the storage medium of the aggregate is suspect, the **ioeagslv** utility must be used to verify or repair the structure of the aggregate.

Use the utility's `-recoveronly`, `-verifyonly`, and `-salvageonly` options to indicate the operations the salvager is to perform on the specified aggregate, as follows:

- Specify the `-recoveryonly` option

  To run recovery on the aggregate without attempting to find or repair any inconsistencies found on it. Recovery is the replaying of the log on the aggregate. Use this option to quickly return consistency to an aggregate that does not need to be salvaged; this represents the normal production use of the salvager. Unless the contents of the log or the physical structure of the aggregate is damaged, replaying the log is an effective guarantee of a file system's integrity.
- Specify the `-verifyonly` option

  To determine whether the structure of the aggregate contains any inconsistencies. Use this option to assess the extent of the damage to an aggregate. The salvager runs log recovery and then determines whether there are any inconsistencies. No repair is attempted other than running log recovery.
- Specify the `-salvageonly` option

  To attempt to repair any inconsistencies that are found in the structure of the aggregate without first running recovery on it. Use this option if you believe the log is damaged or replaying the log does not return consistency to the aggregate and might in fact further damage it. In most cases, you do not salvage an aggregate without first recovering it.
- Omit the `-recoveronly`, `-verifyonly`, and `-salvageonly` options

  To run recovery on the aggregate and then attempt to repair any inconsistencies that are found in the structure of the aggregate. Because recovery eliminates inconsistencies in an undamaged file system, an aggregate is typically recovered before it is salvaged. In general, it is good first to recover and then to salvage an aggregate if a system goes down or experiences a hardware failure.

  Omit these three options if you believe the log should be replayed before attempts are made to repair any inconsistencies that are found on the aggregate. (Omitting the three options is equivalent to specifying the `-recoveronly` and `-salvageonly` options.)

The salvager utility can set or clear the aggregate damaged bit:

- The `-verifyonly` option can set the bit if a true corruption is found or clear it if no corruption is found.
- Repair (with no option) can clear the bit if a successful repair is done.

The following rule summarizes the interaction of the `-recoveronly`, `-verifyonly`, and `-salvageonly` options: The salvage command runs recovery on an aggregate and attempts to repair it unless one of the three salvage options is specified; after one of these options is specified, you must explicitly request any operation that you want the salvager to perform on the aggregate.

The basic function of the salvager is similar to that of the **fsck** program in many UNIX systems. The salvager recovers a zFS aggregate and repairs problems it detects in the structure of the aggregate. It does not verify or repair the format of user data that is contained in files on the aggregate.

The salvager verifies the structure of an aggregate by examining all of the anodes, directories, and other metadata in each file system on the aggregate. An *anode* is an area on the disk that provides information that is used to locate data such as files, directories, ACLs, and other types of file system objects. Each file system contains an arbitrary number of anodes, all of which must reside on the same aggregate. By following the links between the various types of anodes, the salvager can determine whether the organization of an aggregate and the file system it contains is correct and make repairs if necessary.

The salvager is designed to make all repairs in one pass, but due to the nature of the program's inputs (a corrupted, possibly vastly corrupted file system) IBM recommends a second running of the salvage program to verify that the aggregate is truly repaired. If verifying the aggregate shows that it is not repaired, then you should try running the salvager again to repair the aggregate. If this does not repair the aggregate, you can create a copy of the aggregate and run the salvager more times to try to repair it. If the salvager cannot repair the aggregate after several repair attempts, the copy of the aggregate and salvager job logs will allow IBM service to determine why.

Not all aggregates can be salvaged. In cases of extensive damage to the structure of the metadata on an aggregate or damage to the physical disk that houses an aggregate, the salvager cannot repair inconsistencies. Also, the salvager cannot verify or repair damage to user data on an aggregate. The salvager cannot detect problems that modified the contents of a file but did not damage the structure of an aggregate or change the metadata of the aggregate.

Like the **fsck** command, the salvager analyzes the consistency of an aggregate by making successive passes through the aggregate. With each successive pass, the salvager examines and extracts a different type of information from the blocks and anodes on the aggregate. Later passes of the salvager use information that is found in earlier passes to help in the analysis.

It is possible for the salvager to attempt a dynamic grow of an aggregate. One possible reason for this is if an extended (v5) directory is found to be inconsistent (or broken). The salvager will try to repair it by converting it to a new extended (v5) directory. To do this might require more disk space. If the disk space is not available, the directory is marked read-only. The rest of the file system has already been made consistent, so you should still be able to mount the file system and read from the directory.

In general, if the salvager is invoked for a VSAM linear data set that it is sure is not a zFS aggregate, it exits with an error code of at least 16 without analyzing the VSAM linear data set. It exits with an error code of EBUSY (114) if a file system on the aggregate to be recovered or salvaged is mounted or attached. (If necessary, you can use the UNMOUNT command to unmount the aggregate.)

Beginning in z/OS V2R1, the salvager no longer supports salvaging aggregates that contain more than one file system or clones (.bak file systems). For additional details about running the salvage utility, see “Understanding the salvager utility” on page 94.

As the salvager runs, it maintains a list of sorted error records that need repair. Each record includes details for the salvager to quickly repair the aggregate. The salvager displays corruption messages if verification found any inconsistency. It also displays progress messages (IOEZ00782I) during verification to indicate how many objects have been processed. Depending on the aggregate size and system usage, the salvager batch job might take hours or even longer to complete.

For a batch job, the **ioeagslv** options are specified in the EXEC PARM as a single subparameter (a single character string enclosed in apostrophes with no commas separating the options). You cannot put the ending apostrophe in column 72. If it needs go to the next line, use a continuation character in column 72 (continuing in column 16 with the ending apostrophe on the second line). Remember that a JCL EXEC PARM is limited to 100 characters. See the topic on the EXEC PARM in *z/OS MVS JCL Reference*. For an example of the EXEC PARM for **ioeagslv**, see Figure 42 on page 135.

The zFS configuration file can include debugging parameters for the salvager utility. The debugging parameters are described in "IOEFSPRM" on page 220. There are two ways that you can implement the configuration file:
- As a single file that is defined by a IOEZPRM DD card
- As one or more parameter file members, named IOEPRM*xx*

You can provide an optional IOEZPRM DD statement in the JCL for the batch job to specify the location of the IOEFSPRM file. Or, you can omit the IOEZPRM DD statement and specify the `-PRM` option on the EXEC PARM to use IOEPRMxx parameter file members. If you do not specify the IOEZPRM DD statement, the utility searches the logical parmlib concatenation to find the IOEPRM*xx* members that contain the debugging parameters, in the same way that the zFS PFS does if you do not specify the IOEZPRM DD statement in the ZFS PROC. For more information about specifying the configuration file, see "IOEFSPRM" on page 220.

ioeagslv causes the backup change activity flag to be set if the log is replayed or a repair is done.

ioeagslv can be used to salvage aggregate versions 1.4 and 1.5.

Privilege required

The user must meet one of the authorization requirements:
- Have UPDATE authority for the specified VSAM linear data set.
- Be UID 0.
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

If you are using an IOEFSPRM file in your ZFS PROC, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

Figure 42 on page 135 shows an example of a job that invokes the **ioeagslv** utility.

```
//USERIDA  JOB ,'Salvage',
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//SALVAGE  EXEC   PGM=IOEAGSLV,REGION=0M,
// PARM=('-aggregate OMVS.PRV.COMPAT.AGGR001 -verifyonly')
//IOEZPRM DD  DSN=SYS4.PVT.SY1.PARMLIB(IOEFSPRM),DISP=SHR
//SYSPRINT DD     SYSOUT=H
//STDOUT   DD     SYSOUT=H
//STDERR   DD     SYSOUT=H
//SYSUDUMP DD     SYSOUT=H
//CEEDUMP  DD     SYSOUT=H
//*
```

Figure 42. Job to verify a zFS aggregate using debug parameters specified in IOEFSPRM

```
//USERIDA  JOB ,'Salvage',
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//SALVAGE  EXEC   PGM=IOEAGSLV,REGION=0M,
// PARM=('-aggregate OMVS.PRV.COMPAT.AGGR001 -verifyonly -PRM=(03)')
//SYSPRINT DD     SYSOUT=H
//STDOUT   DD     SYSOUT=H
//STDERR   DD     SYSOUT=H
//SYSUDUMP DD     SYSOUT=H
//CEEDUMP  DD     SYSOUT=H
//*
```

Figure 43. Job to verify a zFS aggregate using debug parameters specified in parmlib member IOEPRM03

ioefsutl

Purpose

This section introduces the **ioefsutl** batch utility suite. It is run as a batch job. A zFS aggregate must be unmounted (and not attached) before **ioefsutl** can process it.

ioefsutl is a batch utility that supports the following functions:

- **format** of a new aggregate in the specified version.
- **salvage** to verify and repair a damaged aggregate.
- **converttov5** to change a version 1.4 aggregate to a version 1.5 aggregate and convert all the existing directories to extended (v5) directories.
- **converttov4** to convert all extended (v5) directories to v4 directories and then change the version 1.5 aggregate to a version 1.4 aggregate.

If you are using the IOEFSPRM file, you can provide an optional IOEZPRM DD statement in the JCL for a batch job to specify the location of the IOEFSPRM file. If you are using the IOEPRM*xx* parmlib member, omit the IOEZPRM DD statement and specify the `-PRM` option on the EXEC PARM; for example `-PRM=(03)` if your configuration file is in the parmlib member IOEPRM03. If you do not specify the IOEZPRM DD statement, the utility searches the logical parmlib concatenation to find the IOEPRMxx members that contain the debugging parameters, in the same way that the zFS PFS does if you do not specify the IOEZPRM DD statement in the ZFS PROC. For more information about specifying the configuration file, see "IOEFSPRM" on page 220.

ioefsutl converttov4

Purpose

ioefsutl converttov4 is a batch utility that converts a version 1.5 aggregate to a version 1.4 aggregate.

Format

```
ioefsutl converttov4 -aggregate name [-verbose][-level][-help]
```

Options

-aggregate *name*
Specifies the name of the aggregate to be converted. The aggregate name is not case-sensitive. It is translated to uppercase.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

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

-verbose
Displays starting and ending messages of each directory being converted.

Usage

The **ioefsutl converttov4** command is used when you need to convert a zFS version 1.5 aggregate to a version 1.4 aggregate. All extended (v5) directories are converted to v4 directories. You might use this if you need to run z/OS releases prior to z/OS V2R1.

ioefsutl converttov4 cannot convert the version 1.5 aggregate if it has grown larger than approximately 4 TB. In this case, you must copy subsets of the data one at a time into other version 1.4 aggregates using the z/OS UNIX shell command **pax**. Each subset must be copied into a separate version 1.4 aggregate that is less than 4 TB.

ioefsutl converttov4 cannot convert a directory that contains more than 64K-1 subdirectories. In this case, you must copy subsets of the directory into separate directories contained in a version 1.4 aggregate.

Converting a directory from an extended (v5) directory to a version 4 directory requires both versions of the directory to be on the disk at the same time, temporarily. If the aggregate becomes full during the allocation of the new directory, a dynamic grow is attempted. See "Dynamically growing a compatibility mode aggregate" on page 30 for information about controlling the dynamic growth of an aggregate. If there is not enough space to complete the conversion, the new directory is deleted and the conversion operation fails.

The old directory is deleted when the conversion is completed. The resulting new directory can possibly require more space than the old directory, and could also possibly require less space than the old directory. Results will vary based on the actual directory contents.

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

The conversion will cause the backup change activity flag to be set.

If the aggregate damaged bit is set, conversion does not start and an error is issued.

If the aggregate damaged bit is set, you can still mount the aggregate. The IOEZ00783E console message is displayed:

```
IOEZ00783E Aggregate aggregate_name is damaged
```

Privilege required

The user must meet one of the authorization requirements:

- Have UPDATE authority for the specified VSAM linear data set
- Be UID 0
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class

If you are using an IOEFSPRM file in your JCL, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

Figure 44 shows an example of a job that invokes the **ioefsutl** utility to convert a version 1.5 aggregate to a version 1.4 aggregate.

```
//USERIDA JOB ,'Convert to version 4',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//CONVERT  EXEC PGM=IOEFSUTL,REGION=0M,
// PARM=('converttov4 -aggregate OMVS.PRV.COMPAT.AGGR001')
//SYSPRINT DD SYSOUT=H
//STDOUT   DD SYSOUT=H
//STDERR   DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP  DD SYSOUT=H
//*
```

Figure 44. Job to convert a version 1.5 aggregate to a version 1.4 aggregate

In the PARM=('converttov4 -aggregate OMVS.PRV.COMPAT.AGGR001') statement, the `converttov4` and option `-aggregate` must be in lowercase.

ioefsutl converttov5

Purpose

ioefsutl converttov5 is a batch utility that converts a version 1.4 aggregate to a version 1.5 aggregate.

Format

```
ioefsutl converttov5 -aggregate name -aggrversion_only [-verbose][-level][-help]
```

Options

-aggregate *name*
: Specifies the name of the aggregate to be converted. The aggregate name is not case-sensitive. It is translated to uppercase.

-aggrversion_only
: Only the aggregate version is converted from 1.4 to 1.5. No directories are converted.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

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

-verbose
: Displays starting and ending messages of each directory being converted.

Usage

The **ioefsutl converttov5** command is used when you need to convert a zFS version 1.4 aggregate to a version 1.5 aggregate. All v4 directories are converted to extended (v5) directories. You might use this command if you have migrated all | your systems to z/OS V2R1 or later and you want to exploit extended (v5) directories.

Converting a directory from version 4 to an extended (v5) directory requires both versions of the directory to exist on disk at the same time, temporarily. If the aggregate becomes full during the allocation of the new directory a dynamic grow will be attempted. See “Dynamically growing a compatibility mode aggregate” on page 30 for information about controlling dynamic growth of an aggregate. If there is not enough space to complete the conversion, the new directory is deleted and the conversion operation fails.

When the conversion is completed, the old directory is deleted. The resulting new directory can possibly require more space than the old directory, and could also possibly require less space than the old directory. Results will vary based on the actual directory contents.

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

The conversion causes the backup change activity flag to be set.

If the aggregate damaged bit is set, conversion does not start and an error is issued.

If the aggregate damaged bit is set, you can still mount the aggregate. The IOEZ00783E console message is displayed:

```
IOEZ00783E Aggregate aggregate_name is damaged
```

Privilege required

The user must meet one of the authorization requirements:

- Have UPDATE authority for the specified VSAM linear data set.
- Be UID 0.
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

If you are using an IOEFSPRM file in your JCL, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

Figure 45 shows an example of a job that invokes the **ioefsutl** utility to convert a version 1.4 aggregate to a version 1.5 aggregate.

```
//USERIDA JOB ,'Convert to version 5',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//CONVERT EXEC PGM=IOEFSUTL,REGION=0M,
// PARM=('converttov5 -aggregate OMVS.PRV.COMPAT.AGGR001')
//SYSPRINT DD SYSOUT=H
//STDOUT   DD SYSOUT=H
//STDERR   DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP  DD SYSOUT=H
//*
```

Figure 45. Job to convert a version 1.4 aggregate to a version 1.5 aggregate

In the `PARM=('converttov5 -aggregate OMVS.PRV.COMPAT.AGGR001')` statement, the `converttov5` and option `-aggregate` must be in lowercase.

ioefsutl format

Purpose

| **ioefsutl format** is a batch utility that formats a VSAM linear data set to become a
| version 4 or version 5 zFS compatibility mode aggregate.

Format

```
ioefsutl format -aggregate name [-size blocks] [-logsize blocks] [-owner uid|name]
[-group gid|name] [-perms number] [-grow blocks][-overwrite][{-version4 | -version5}]
 [-level][-help]
```

Options

-aggregate *name*
Specifies the name of the data set to format. This is also the aggregate name. The aggregate name is always translated to uppercase and cannot be longer than 44 characters. The following characters can be included in the name of an aggregate:
- All uppercase and lowercase alphabetic characters (a to z, A to Z)
- All numerals (0 to 9)
- The . (period)
- The - (dash)
- The _ (underscore)
- The @ (at sign)
- The # (number sign)
- The $ (dollar).

-group *gid* | *name*
Specifies the group owner for the root directory of the file system. It can be specified as a z/OS group name or as a GID. The default is the GID of the issuer of **ioefsutl format**. If only `-owner` *name* is specified, the group is that owner's default group. If only `-owner` *uid* is specified, the group is the issuer's group.

-grow *blocks*
Specifies the number of 8-KB blocks that zFS uses as the increment for extension when the `-size` option specifies a size greater than the primary allocation.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

-logsize *blocks*
Specifies the size in 8-KB blocks of the log. The valid range is from 13 to 16384 blocks (128 megabytes). The default is 1% of the aggregate size. This default logsize will never be smaller than 14 blocks and it will never be larger than 4096 blocks (32 megabytes). This size is normally sufficient. However, a small aggregate that is grown to be very large will still have a small log. You might want to specify a larger log if you expect the aggregate to grow very large.

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

-overwrite
Required if you are reformatting an existing aggregate. Use this option with caution because it destroys any existing data. This option is not usually specified.

-owner *uid* | *name*
Specifies the owner for the root directory of the file system. It can be specified as a z/OS user ID or as a UID. The default is the UID of the issuer of **ioefsutl format**.

-perms *number*
Specifies the permissions for the root directory of the file system. The number can be specified as octal (for example, o755), as hexadecimal (for example, x1ED), or as decimal (for example, 493). The default is o755 (owner read/write-execute, group read-execute, other read-execute).

-size *blocks*
Specifies the number of 8-KB blocks that should be formatted to form the zFS aggregate. The default is the number of blocks that will fit in the primary allocation of the VSAM linear data set. If a number less than the default is specified, it is rounded up to the default. If a number greater than the default is specified, a single extend of the VSAM linear data set is attempted after the primary allocation is formatted unless the `-grow` option is specified. In that case, multiple extensions of the amount that is specified in the `-grow` option will be attempted until the `-size` is satisfied. The size can be rounded up to a control area (CA) boundary by DFSMS. It is not necessary to specify a secondary allocation size on the DEFINE of the VSAM linear data set for this extension to occur. Space must be available on the volume.

-version4
Specifies that the aggregate is to be formatted as a version 1.4 aggregate. See the "Usage notes for ioefsutl format" for information about how the default aggregate version is determined.

-version5
Specifies that the aggregate is to be formatted as a version 1.5 aggregate. See "Usage notes for ioefsutl format" for information about how the default aggregate version is determined. IBM recommends that you do not
| use `-version5` until all your systems are at z/OS V2R1 or later.

Usage notes for ioefsutl format

| The **`ioefsutl format`** utility formats an existing VSAM linear data set as a zFS
| aggregate. All zFS aggregates must be formatted before use.

The aggregate name is not case-sensitive. It is translated to uppercase. If `-version4` or `-version5` is specified, you can run **`ioefsutl format`** even if the zFS PFS is not active on the system. If neither is specified, the aggregate version default is determined by a call to the zFS PFS to obtain the value of the *format_aggrversion* variable from the IOEFSPRM file. If the zFS PFS is not active, then the format will fail.

The size of the aggregate is as many 8 K blocks as fits in the primary allocation of the VSAM linear data set or as specified in the `-size` option. The `-size` option can cause one additional extension to occur during formatting. To extend it further, use the **`zfsadm grow`** command. If `-overwrite` is specified, all existing primary and secondary allocations are formatted and the size includes all of that space. If

`-overwrite` is specified, the backup change activity flag is set. If the VSAM linear data set has a SHAREOPTIONS value of other than 3, **`ioefsutl format`** changes it to SHAREOPTIONS 3 during format.

For a batch job, the **`ioefsutl format`** options are specified in the EXEC PARM as a single subparameter (a single character string enclosed in apostrophes with no commas separating the options). You cannot put the ending apostrophe in column 72. If it needs go to the next line, use a continuation character in column 72 (continuing in column 16 with the ending apostrophe on the second line). Remember that a JCL EXEC PARM is limited to 100 characters. See the topic on EXEC PARM in *z/OS MVS JCL Reference*.

`ioefsutl format` always formats with a unique auditfid.

Privilege required

The user must meet one of the following authorization requirements:
- Have ALTER authority to the VSAM linear data set.
- Be UID 0.
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

UPDATE authority to the VSAM linear data set is sufficient for format, but zFS will not be able to set the zFS bit in the catalog unless the issuer has ALTER authority.

If you are changing the owner or group to something other than the issuer or you are changing the permissions to other than the default, you need UID 0 or READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

If you are using an IOEFSPRM file in your ZFS PROC, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Restrictions

The zFS aggregate cannot be mounted (or attached). The batch job must be issued from a V2R1 or later system and the VSAM linear data set must exist. If neither `-version4` nor `-version5` are specified, the value of the `format_aggrversion` parameter on the server is used. In this case, if the value of the FORMAT_AGGRVERSION parameter cannot be determined, the format will fail.

Examples

Figure 46 on page 144 shows an example of a job that creates and formats a version 1.4 aggregate.

```
//USERIDA JOB ,'Compatibility Mode',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//AMSDUMP DD SYSOUT=H
//DASD0 DD DISP=OLD,UNIT=3390,VOL=SER=PRV000
//SYSIN DD *
     DEFINE CLUSTER (NAME(OMVS.PRV.COMPAT.AGGR001) -
            VOLUMES(PRV000) -
            LINEAR CYL(25 0) SHAREOPTIONS(3))
/*
//CREATE EXEC PGM=IOEFSUTL,REGION=0M,
// PARM=('format -aggregate OMVS.PRV.COMPAT.AGGR001 -version4')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
//*
```

Figure 46. Sample job to create and format a version 1.4 aggregate

Note: In the `PARM=('format -aggregate OMVS.PRV.COMPAT.AGGR001 -version4')` statement, the format, and options `-aggregate` and `-version4` must be in lowercase.

ioefsutl salvage

Purpose

`ioefsutl salvage` is a batch utility that scans an aggregate and reports inconsistencies. Aggregates can be verified, recovered (that is, the log is replayed), or salvaged (that is, the aggregate is repaired). This utility is known as the salvager.

This utility is not normally needed. If a system failure occurs, the aggregate log is replayed automatically the next time the aggregate is attached or mounted. This action typically brings the aggregate back to a consistent state. The aggregate must not be mounted (or attached) when **`ioefsutl salvage`** is run.

Format

```
ioefsutl salvage -aggregate name [-verifyonly][-level][-help]
```

Options

-aggregate *name*
: Specifies the name of the aggregate to be verified or salvaged. The aggregate name is not case-sensitive. It is translated to uppercase.

-help
: Prints the online help for this command. All other valid options that are specified with this option are ignored.

-level
: Prints the level of the **`ioefsutl`** command. This information is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

-verifyonly
: Specifies that the salvager is to verify the specified aggregate. It should not attempt to repair any damage found. The log is replayed before the verification unless an error occurs during the replay. If this option is omitted, the salvager will replay the log, verify the specified aggregate, and then attempt to repair any damage that was found.

Results

For `-verifyonly`, the salvager returns the following return codes:

00
: Success. The aggregate is correct and no repair is needed.

04
: The aggregate has some inconsistencies that need repair.

08
: An error occurred during verification; the report might be incomplete.

12
: A severe error occurred during verification. Verify that processing was halted. The aggregate is not repairable.

16
: Terminating error.

EIO
: The salvager could not read or write the DASD.

EBUSY
: The aggregate was mounted or attached.

EMVSERR
: The salvager had an internal error. This return code is preceded by a dump for an abend 2C3 and reason code EA660701.

ENOMEM
The salvager ran out of storage.

EINVAL
The salvager arguments were incorrect.

For no options specified, the salvager returns the following return codes:

00 Success. The aggregate is correct and no repair is needed.

04 The aggregate had some inconsistencies that were repaired.

08 An error occurred during verification; the report might be incomplete; the aggregate could not be repaired.

12 A severe error occurred during verification; verify processing halted; the aggregate could not be repaired.

16 Terminating error.

EIO The salvager could not read or write the DASD.

EBUSY
The aggregate was mounted or attached.

EMVSERR
The salvager had an internal error. This return code is preceded by a dump for an abend 2C3 and reason code EA660701.

ENOMEM
The salvager ran out of storage.

EINVAL
The salvager arguments were incorrect.

Usage

You can run **`ioefsutl salvage`** even if the zFS PFS is not active on the system. The **`ioefsutl salvage`** utility invokes the salvager on the zFS aggregate that is specified with the `-aggregate` option.

The salvager cannot process an aggregate that contains multiple file systems or a clone.

The processing of the aggregate is controlled by the specification or the omission of the `-verifyonly` option.

- Specify the `-verifyonly` option

  To determine whether the structure of the aggregate contains any inconsistencies. Use this option to assess the extent of the damage to an aggregate. The salvager runs log recovery and then determines whether there are any inconsistencies. No repair is attempted other than running log recovery.
- Omit the `-verifyonly` option

  To run log recovery on the aggregate, verify the aggregate and then attempt to repair any inconsistencies that are found in the structure of the aggregate. Because log recovery eliminates inconsistencies in an undamaged file system, an aggregate is typically recovered before it is salvaged. In general, it is good practice to first recover and then to salvage an aggregate if a system goes down or experiences a hardware failure.

The salvager sets the backup change activity flag if log recovery is run or a repair is done.

The basic function of the salvager is similar to that of the **fsck** program in many UNIX systems. The salvager recovers a zFS aggregate and repairs problems it detects in the structure of the aggregate. It does not verify or repair the format of user data that is contained in files on the aggregate.

The salvager verifies the structure of an aggregate by examining all of the anodes, directories, and other metadata in each file system on the aggregate. An *anode* is an area on the disk that provides information that is used to locate data such as files, directories, ACLs, and other types of file system objects. Each file system contains an arbitrary number of anodes, all of which must reside on the same aggregate. By following the links between the various types of anodes, the salvager can determine whether the organization of an aggregate and the file system that it contains is correct and make repairs if necessary.

Not all aggregates can be salvaged. In cases of extensive damage to the structure of the metadata on an aggregate or damage to the physical disk that houses an aggregate, the salvager cannot repair inconsistencies. Also, the salvager cannot verify or repair damage to user data on an aggregate. The salvager cannot detect problems that modified the contents of a file but did not damage the structure of an aggregate or change the metadata of the aggregate.

The salvager is designed to make all repairs in one pass, but due to the nature of the program's inputs (a corrupted, possibly vastly corrupted file system) IBM recommends a second running of the salvage program to verify that the aggregate is truly repaired. If verifying the aggregate shows that it is not repaired, then you should try running the salvager again to repair the aggregate. If this does not repair the aggregate, you can create a copy of the aggregate and run the salvager more times to try to repair it. If the salvager cannot repair the aggregate after several repair attempts, the copy of the aggregate and salvager job logs will allow IBM service to determine why.

Like the **fsck** command, the salvager analyzes the consistency of an aggregate by making successive passes through the aggregate. With each successive pass, the salvager examines and extracts a different type of information from the blocks and anodes on the aggregate. Later passes of the salvager use information that was found in earlier passes to help in the analysis.

It is possible for the salvager to attempt a dynamic grow of an aggregate. One possible reason for this is if an extended (v5) directory is found to be inconsistent (or broken). The salvager will try to repair it by converting it to a new extended (v5) directory. To do this might require more disk space. If the disk space is not available the directory is marked read-only. The rest of the file system has already been made consistent, so you should still be able to mount the file system and read from the directory.

In general, if the salvager is invoked for a VSAM linear data set that it is sure is not a zFS aggregate, it exits with an error code of at least 16 without analyzing the VSAM linear data set. It exits with an error code of EBUSY (114) if a file system on the aggregate to be recovered or salvaged is mounted or attached. (If necessary, you can use the **unmount** command to unmount the aggregate.)

As the salvager runs, it maintains a list of sorted error records that need repair. Each record includes details for the salvager to quickly repair the aggregate. The salvager displays corruption messages if verification found any inconsistencies. It also displays progress messages (IOEZ00782I) during verification to indicate how many objects were processed. Depending on the aggregate size and system usage, the salvager batch job may take hours or even longer to complete.

For more information about running the salvage utility, see "Understanding the salvager utility" on page 94.

For a batch job, the **`ioefsutl salvage`** options are specified in the EXEC PARM as a single subparameter (a single character string enclosed in apostrophes with no commas separating the options). You cannot put the ending apostrophe in column 72. If it needs to go to the next line, use a continuation character in column 72 (continuing in column 16 with the ending apostrophe on the second line). Remember that a JCL EXEC PARM is limited to 100 characters. See the topic on the EXEC PARM in *z/OS MVS JCL Reference*. For an example of the EXEC PARM for **`ioefsutl salvage`**, see Figure 47.

`ioefsutl salvage` can be used to salvage aggregate versions 1.4 and 1.5.

The salvager utility can set or clear the aggregate damaged bit:
- The `-verifyonly` option can set the bit if a true corruption is found or clear it if no corruption is found.
- Repair (with no option) can clear the bit if a successful repair is done.

Privilege required

The user must meet one of the following authorization requirements:
- Have UPDATE authority to the VSAM linear data set
- Be UID 0
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class

If you are using an IOEFSPRM file in the JCL, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

Figure 47 shows an example of a job to salvage a zFS aggregate:

```
//USERIDA JOB ,'Salvage verify',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//SALVAGE  EXEC PGM=IOEFSUTL,REGION=0M,
// PARM=('salvage -aggregate OMVS.PRV.COMPAT.AGGR001 -verifyonly')
//IOEZPRM DD DSN=SYS4.PVT.SY1.PARMLIB(IOEFSPRM),DISP=SHR
//SYSPRINT DD SYSOUT=H
//STDOUT   DD SYSOUT=H
//STDERR   DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP  DD SYSOUT=H
//*
```

Figure 47. Job to verify a zFS aggregate using debug parameters specified in IOEZPRM

Note: In the `PARM=('salvage -aggregate OMVS.PRV.COMPAT.AGGR001 -verifyonly')` statement, the salvage and options `-aggregate` and `-verifyonly` must be in lowercase.

MOUNT

Purpose

MOUNT is a TSO/E command that mounts a file system into the z/OS UNIX hierarchy. This section only documents MOUNT options that are unique to zFS. It can also be invoked from the z/OS UNIX shell (`/usr/sbin/mount`). For additional information about this command, see *z/OS UNIX System Services Command Reference*.

Note:

1. Beginning with z/OS V2R1, zFS clones are no longer supported. An attempt to mount an aggregate that contains a .bak (clone) file system will be denied.
2. Beginning with z/OS V2R1, multi-file system aggregates are no longer supported. An attempt to mount a zFS file system that is contained in a zFS multi-file system aggregate is denied.

Format

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

Options

TYPE (*file_system_type*)
: Specifies the file system type. Specify ZFS or HFS and the correct file system type is determined for the file system that is located by the data set name. If the TYPE specified (HFS) does not match the real file system type (ZFS), any associated ZFS parameters are ignored. For more information, see Mounting considerations in *z/OS UNIX System Services Planning*.

PARM(*parameter_string*)
: Specifies a parameter string to be passed to zFS. Parameters are case-sensitive and separated by a comma. Enclose the parameter string within quotation marks. If a parameter is specified multiple times, the last parameter is used.

Note: If the value specified on the TYPE parameter (HFS) does not match the real file system type (ZFS), any associated ZFS parameters are ignored.

AGGRFULL(*threshold*,*increment*)
: Specifies the threshold and increment for reporting aggregate full error messages to the operator. The default is the `aggrfull` specification in the IOEFSPRM file.

Note: AGGRFULL and FSFULL provide the same function. You can use either one (or both) to monitor the free space for an aggregate. However, AGGRFULL tends to give a more accurate view of free space and is the suggested choice.

AGGRGROW | NOAGGRGROW
: Specifies whether the aggregate is eligible to be dynamically grown. The growth is based on the secondary allocation of the aggregate and will occur when the aggregate becomes full. The default is the `aggrgrow` specification in the IOEFSPRM file.

CONVERTTOV5 | NOCONVERTTOV5
: Specifies whether a zFS read/write file system is assigned the

converttov5 attribute. If it is assigned the converttov5 attribute and the aggregate is a version 1.5 aggregate, zFS automatically converts directories from v4 to extended (v5) as they are accessed. If the converttov5 attribute is assigned at primary mount time, a version 1.4 aggregate is changed to a version 1.5 aggregate.

If automatic directory conversion for a directory fails, the conversion is not attempted again until the file system is unmounted and mounted again.

The converttov5 attribute can also be assigned if the MOUNT option is not specified but the converttov5 specification in the IOEFSPRM file is on when the file system is mounted or remounted.

The default is NOCONVERTTOV5. However, the converttov5 attribute can also be assigned if the converttov5 specification in the IOEFSPRM file is on when the file system is mounted or remounted.

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

Note: AGGRFULL and FSFULL provide the same function. You can use either one (or both) to monitor the free space for an aggregate. However, AGGRFULL tends to give a more accurate view of free space and is the suggested choice.

RWSHARE | NORWSHARE
Specifies whether a zFS read/write mounted file system will be mounted sysplex-aware or non-sysplex aware. zFS must be running sysplex-aware on a file system basis (IOEFSPRM specifies *sysplex=filesys*) for this parameter to take effect. The default is the sysplex_filesys_sharemode specified in the IOEFSPRM file, or later using the **zfsadm config** command. For information about whether to make a read/write file system sysplex aware, see "Using zFS read/write sysplex-aware file systems" on page 17.

Usage

A mount of a compatibility mode aggregate is serialized with other **zfsadm** commands (because the mount of a compatibility mode aggregate does an implicit attach).

If you attempt to mount a compatibility mode aggregate/file system read-only and it fails because it needs to run recovery (return code EROFS (141) and reason code EF*xx*6271), you should temporarily mount it read/write (so it can complete the recovery process) and then mount it read-only. Alternatively, you can specify the romount_recovery=on configuration option in IOEFSPRM. This causes the file system to automatically be temporarily mounted read/write to allow log recovery to run and then to be mounted read-only.

If the DASD volume containing the zFS compatibility mode aggregate being mounted is read-only, you can receive message IOEZ00336I. This message indicates that the zFS aggregate indicator cannot be set in the catalog (actually, in the VVDS on the volume). The zFS aggregate is successfully mounted (and attached).

DFSMSdss backup (DUMP) will not automatically quiesce and unquiesce the zFS aggregate because it cannot determine that the VSAM linear data set is a zFS aggregate. If the zFS aggregate can be mounted with the DASD volume in read/write, the zFS aggregate indicator will be set.

You can determine if the zFS aggregate indicator is set by using IDCAMS LISTCAT ALL against the zFS aggregate and looking for the zFS indicator in the output.

Do not use a path entry as the file system name in the MOUNT command (see the topic on DEFINE PATH in *z/OS DFSMS Access Method Services Commands*). The mount succeeds but the system issues messages similar to the following:

```
IOEZ00412I Catalog search failed for aggregate PLEX.JMS.AGGR006.PATH. Shareoptions are not altered.

IOEZ00336I PLEX.JMS.AGGR006.PATH could not be marked as a zFS aggregate in the catalog, rc=60 rsn=104
```

Examples

The following TSO/E example mounts a zFS file system and specifies a threshold and increment to display a message when the file system becomes almost full:

```
MOUNT FILESYSTEM('OMVS.PRV.AGGR004.LDS0004') MOUNTPOINT('/etc/zfscompat1')
      TYPE(ZFS) MODE(RDWR) PARM('AGGRFULL(90,5)')
```

Here is the same example as a z/OS UNIX command:

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

Related information

Command:
  UNMOUNT (For information about this command, see *z/OS UNIX System Services Command Reference*.)

File:
  IOEFSPRM

zfsadm

Purpose

This section introduces the **zfsadm** command suite. The **zfsadm** command is run from the z/OS UNIX shell. It can also be invoked from TSO/E by using the program name IOEZADM or as a batch job by using PGM=IOEZADM. If PARM is coded in the JCL to pass options or arguments to IOEZADM and any of the options or arguments contain a slash (for example, R/O), you must specify a leading slash as the first character in the PARM string. See Figure 49 on page 164 for an example of invoking IOEZADM from a batch job.

Command syntax

The **zfsadm** commands have the same general structure:

```
command {-option1 argument... | -option2 {argument1 | argument2}...}
[-optional_information]
```

The following example illustrates the elements of a **zfsadm** command:

zfsadm detach {-all | -aggregate *name*} [-help]

The following list summarizes the elements of the **zfsadm** command:

Command
: A command consists of the command suite (**zfsadm** in the previous example) and the command name (**detach**). The command suite and the command name must be separated by a space. The command suite specifies the group of related commands.

Options
: Command options always appear in monospace type in the text, are always preceded by a **-** (dash), and are often followed by arguments. In the previous example, `-aggregate` is an option, with *name* as its argument. An option and its arguments tell the program which entities to manipulate when running the command (for example, which aggregate, or which file system). In general, the issuer should provide the options for a command in the order detailed in the format description. The { | } (braces separated by a vertical bar) indicate that the issuer must enter either one option or the other (`-all` or `-aggregate` in the previous example).

  Command options are described in alphabetic order to make them easier to locate; this does not reflect the format of the command. The formats are presented the same as on your system.

Arguments
: Arguments for options are *highlighted* in the text. The { | } indicate that the issuer must enter either one argument or the other (`-all` or `-aggregate` in the preceding example). The ... (ellipsis) indicates that the issuer can enter multiple arguments.

Options
: Some commands have optional, as well as required, options, and arguments. Optional information is enclosed in [ ] (brackets). All options except `-all` or `-aggregate` in the previous example are optional.

Options

The following options are used with many **zfsadm** commands. They are also listed with the commands that use them.

-aggregate *name*
: Specifies the aggregate name of the aggregate to use with the command.

-filesystem *name*
: Specifies the file system to use with the command.

-help
: Prints the online help for this command. All other valid options that are specified with this option are ignored. For complete details about receiving help, see "Receiving help" on page 156.

-size *kbytes*
: Specifies the size in K-bytes for the *kbytes* argument.

-system *sysname*
: Specifies the name of the system that the request is sent to.

When an option is specified multiple times on one command, the first one is honored and the subsequent ones are 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 that are used by system administrators to manage file systems and aggregates. You can issue commands from OMVS, TSO/E, or as a batch job. Use the IOEZADM format for TSO/E and batch. For an example, see Figure 49 on page 164. The description of the **zfsadm attach** command shows an example of issuing them as a batch job. The other **zfsadm** commands can be run as a batch job in a similar manner.

For a batch job, the **zfsadm** options are specified in the EXEC PARM as a single subparameter (a single character string enclosed in apostrophes with no commas separating the options). You cannot put the ending apostrophe in column 72. If it needs to go to the next line, use a continuation character in column 72 (continuing in column 16 with the ending apostrophe on the second line). Remember that a JCL EXEC PARM is limited to 100 characters. See the topic on the EXEC PARM in *z/OS MVS JCL Reference*.

zfsadm commands are serialized with each other. That is, when a **zfsadm** command is in progress, a subsequent **zfsadm** command is delayed until the active **zfsadm** completes. This also includes MOUNT of a compatibility mode aggregate (because an implicit attach occurs). This does not include **zfsadm grow** or implicit aggregate grow. **zfsadm** commands do not delay normal file system activity (except when the **zfsadm** command requires it, such as **zfsadm quiesce**).

zfsadm commands only work on zFS file systems and aggregates. All **zfsadm** commands work across sysplex members that are in a shared file system environment.

When supplying an argument to a **zfsadm** command, the option (for example **-aggregate**) associated with the argument (for example, OMVS.PRV.AGGR001.LDS0001) can be omitted if:

- All arguments that are supplied with the command are entered in the order in which they appear in the command's syntax. (The syntax for each command is provided.)
- Arguments are supplied for all options that precede the option to be omitted.
- All options that precede the option to be omitted accept only a single argument.
- No options, either those that accept an argument or those that do not, are supplied before the option to be omitted.
- The first option cannot be followed by an additional option before the vertical bar.

In the case where two options are presented in

```
{ | }
```

(braces separated by a vertical bar), the option associated with the first argument can be omitted if that argument is provided; however, the option associated with the second argument is required if that argument is provided.

If it must be specified, an option can be abbreviated to the shortest possible form that distinguishes it from other options of the command. For example, the `-aggregate` option found in many **zfsadm** commands can typically be omitted or abbreviated to be simply `-a`. (One exception is the **zfsadm attach** command because it has an `-aggrfull` option.)

It is also valid to abbreviate a command name to the shortest form that still distinguishes it from the other command names in the suite. For example, it is acceptable to shorten the **zfsadm grow** command to **zfsadm g** because no other command names in the **zfsadm** command suite begin with the letter **g**. However, there are two **zfsadm** commands that begin with **l**: **zfsadm lsaggr** and **zfsadm lsfs**. To remain unambiguous, they can be abbreviated to **zfsadm lsa** and **zfsadm lsf**.

The following examples illustrate three acceptable ways to enter the same **zfsadm grow** command:

- Complete command:
  ```
  zfsadm grow -aggregate omvs.prv.aggr001.lds0001 -size 50000
  ```
- Abbreviated command name and abbreviated options:
  ```
  zfsadm g -a omvs.prv.aggr001.lds0001 -s 50000
  ```
- Abbreviated command name and omitted options:
  ```
  zfsadm g omvs.prv.aggr001.lds0001 50000
  ```

The ability to abbreviate or omit options is intended for interactive use. If you embed commands in a shell script, do not omit options nor abbreviate them. If an option is added to a command in the future, it might increase the minimum unique abbreviation that is 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 canceled, but the worker thread continues to process the request to completion. In addition, most **zfsadm** commands require a common **zfsadm** lock while they are processing. If the **zfsadm** command cannot get the lock, it waits for it to become available. This means, if you issue another **zfsadm** command (after canceling a previous one), it can be delayed by this common **zfsadm** lock until the previous (possibly canceled) command completes. The **zfsadm fsinfo** command does not have either of these possible processing delays.

Receiving help

There are several different ways to receive help about **zfsadm** commands. The following examples summarize the syntax for the different help options available:

zfsadm help
Displays a list of commands in a command suite.

zfsadm help -topic *command*
Displays the syntax for one or more commands.

zfsadm apropos -topic *string*
Displays a short description of any commands that match the specified *string*.

As of z/OS V1R13, when the **zfsadm** command displays help text or a syntax error message, it will show the name of the command as IOEZADM, instead of **zfsadm**. This occurs because the **zfsadm** command is not a binary module in the z/OS UNIX file system; rather, it is a shell script that invokes IOEZADM. IOEZADM is an entry that has the sticky bit on in the permissions. The sticky bit means that the IOEZADM module is found and executed from the user's STEPLIB, link pack area, or link list concatenation. (IOEZADM is usually located in SYS1.SIEALNKE.) However, you cannot run IOEZADM from the shell because IOEZADM is not normally in your PATH.

Privilege required

zfsadm commands that query information (for example, `lsfs`, `aggrinfo`) require the issuer to have READ authority to the data set that contains the IOEFSPRM file if you are using an IOEFSPRM file in your startup proc, or require no special authorization if you are using parmlib (IOEPRMxx). **zfsadm** commands that modify (for example **format**) additionally require that the issuer must have one of the following:

- UID of 0. If you are permitted READ to the BPX.SUPERUSER resource in the RACF FACILITY class, you can become a UID of 0 by issuing the **su** command.
- READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Specific privilege information is listed within each command’s description.

Related information

Commands:
zfsadm aggrinfo
zfsadm apropos
zfsadm attach
zfsadm config
zfsadm configquery
zfsadm convert
zfsadm define
zfsadm delete
zfsadm detach
zfsadm fileinfo
zfsadm format

```
zfsadm grow
zfsadm help
zfsadm lsaggr
zfsadm lsfs
zfsadm lssys
zfsadm query
zfsadm quiesce
zfsadm setauditfid
zfsadm unquiesce
```

File:
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 sysname] [-fast | -long]
 [-level] [-help]
```

Options

-aggregate *name*
: Specifies the name of an aggregate about which information is to be displayed. The aggregate must be attached. The aggregate name is not case-sensitive. It is translated to uppercase. If this option is omitted, information is provided about all of the attached aggregates on the system. Compatibility mode aggregates are implicitly attached when they are mounted.

-fast Causes the command to display a single line of output for each attached aggregate. See "Usage" on page 159 for an explanation of the information that is displayed on each line.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

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

-long Causes the output of the command to be extended to display the following additional information about space usage in an aggregate:

- Version of the aggregate
- File system identification (auditfid)
- Indicates sysplex-aware when the aggregate is sysplex-aware for read/write
- Indicates converttov5 if the aggregate has the converttov5 attribute
- Number of free 8-KB blocks
- Number of free 1-KB fragments
- Size of the log file
- Size of the filesystem table
- Size of the bitmap file
- If the aggregate is quiesced, the job name, system name and the time stamp of when the quiesce occurred.

-system *sysname*
: Specifies the name of the system that owns the attached aggregates for which the information is displayed.

Usage

The **zfsadm aggrinfo** command lists information about the total amount of disk space and the amount of disk space currently available on attached aggregates. The -aggregate option can be used to specify a single aggregate about which information is to be displayed. If this option is omitted, information about all aggregates that are attached in the sysplex (if shared file systems are being used) or the system is displayed. In a shared file system environment, you can limit the display to a single system by using the -system option. Compatibility mode aggregates are implicitly attached when they are mounted.

This command displays a separate line for each aggregate. Each line displays the following information:
- The aggregate name.
- Whether the aggregate is read/write (R/W) or read-only (R/O), it is a mounted compatibility mode aggregate (COMP) or an attached compatibility mode aggregate (MULT), or the aggregate is currently quiesced (QUIESCED), disabled (DISABLED), or both.
- The amount of space available in KB.
- The total amount of space in the aggregate in KB. (To grow an aggregate using the **zfsadm grow** command, specify a number larger than this number.)
- If -long is specified, the version of the aggregate, the auditfid, sysplex-aware if the aggregate is sysplex-aware for read/write, the converttov5 attribute, the number of free 8-KB blocks, the number of free 1-KB fragments, the size of the log file, the size of the file system table, the size of the bitmap file, and if the aggregate is quiesced, the job name, timestamp, and system name of the job.

Privilege required

| If you are using an IOEFSPRM file in your startup proc, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Example

Following is an example command that displays information about the disk space that is available on all aggregates that are attached in the sysplex.

```
DCEIMGKC:/DCEIMGKC/home/suimgkc> zfsadm aggrinfo -long
IOEZ00369I A total of 1 aggregates are attached to the sysplex.
PLEX.AGGR (R/W COMP QUIESCED): 559 K free out of total 720
version 1.5
auditfid C3C6C3F0 F0F3000E 0000
sysplex-aware, converttov5
           69 free 8k blocks;          7 free 1K fragments
          112 K log file;             16 K filesystem table
            8 K bitmap file
Quiesced by job SUIMGKC3 on system DCEIMGKC on Mon Feb 11 16:04:36 2013
```

Figure 48. Sample of zfsadm aggrinfo -long command

Related information

Commands:
| **zfsadm fsinfo**
zfsadm lsaggr

Files:
IOEFSPRM

zfsadm apropos

Purpose

Shows each help entry containing a specified string.

Format

```
zfsadm apropos -topic string [-level] [-help]
```

Options

-help Prints the online help for this command. All other valid options specified with this option are ignored.

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

-topic Specifies the keyword string for which to search. If it is more than a single word, surround it with quotation marks ("") or another delimiter. Type all strings for **zfsadm** commands in all lowercase letters.

Usage

The **zfsadm apropos** command displays the first line of the online help entry for any **zfsadm** command containing the string specified by `-topic` in its name or short description. To display the syntax for a command, use the **zfsadm help** command.

Privilege required

| If you are using an IOEFSPRM file in your startup 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
```

Related information

Commands:
zfsadm help

zfsadm attach

Purpose

Attaches an aggregate to zFS without mounting the file system. Beginning in z/OS V2R2, this aggregate can only contain one file system.

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

Format

```
zfsadm attach {-aggregate name [-system sysname]}
[-aggrfull threshold,increment]
[{-R/O | -ro | -rw}] [-nbs | -nonbs] [-aggrgrow | -noaggrgrow][-level] [-help]
```

Options

-aggregate *name*
Specifies the name of the aggregate to be attached. The aggregate name is not case-sensitive. It is translated to uppercase. This aggregate does not need an entry in the IOEFSPRM file.

Compatibility mode aggregates do not need to be attached with the **zfsadm attach** command. They are automatically attached on MOUNT of the compatibility mode file system.

-aggrfull *threshold,increment*
Specifies the threshold and increment for reporting aggregate full error messages to the operator. Both numbers must be specified. The first number is the threshold percentage and the second number is the increment percentage. For example, if 90,5 were specified, the operator is notified when the aggregate is 90% full, then again at 95% full, and again at 100% full. The default is the global `aggrfull` entry of the IOEFSPRM file.

-aggrgrow
Specifies that the aggregate should be dynamically grown if it runs out of physical space. The aggregate (that is, the VSAM linear data set) must have a secondary allocation specified and there must be space available on the volume. The default is the `aggrgrow` option of the IOEFSPRM file.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

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

-nbs Specifies that new block security is used for file systems in this aggregate. *New block security* refers to the guarantee made when a system fails. If a file was being extended or new blocks were being allocated for the file, but the user data had not yet made it to the disk when the failure occurred, zFS shows the newly allocated blocks as all binary zeros and not whatever was on disk in those blocks at time of failure.

-nonbs
The NONBS option is no longer supported; if NONBS is specified, it is ignored. zFS always runs with NBS on.

-noaggrgrow
: Specifies that the aggregate should not be dynamically grown if it runs out of physical space. The default is the `aggrgrow` option of the IOEFSPRM file.

-R/O | -ro
: Specifies that the aggregate should be opened in read-only mode. The default is read/write unless `-R/O or -ro` is specified.

-rw
: Specifies that the aggregate should be opened in read/write mode. The default is read/write unless `-R/O or -ro` is specified.

-system *sysname*
: Specifies the name of the system that will be the zFS owner of the aggregate. The system name is not case-sensitive. It is translated to uppercase.

Usage

| The **`zfsadm attach`** command attaches zFS aggregates on this system. Beginning in
| z/OS V2R2, zFS only attaches aggregates that contain exactly one file system.

If the attach fails because log recovery is unsuccessful, you can run the **`ioefsutl salvage`** batch utility with the `-verifyonly` option on the aggregate to determine if there is an inconsistency. If so, use **`ioefsutl salvage`** to recover the aggregate and reissue the **`zfsadm attach`** command.

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

If the DASD volume containing the zFS aggregate that being attached is read-only, you might receive message IOEZ00336I. This indicates that the zFS aggregate indicator cannot be set in the catalog (actually, in the VVDS on the volume). The zFS aggregate is successfully attached. DFSMSdss backup (DUMP) will not automatically quiesce and unquiesce the zFS aggregate because it cannot determine that the VSAM linear data set is a zFS aggregate. If the zFS aggregate can be attached with the DASD volume in read/write, the zFS aggregate indicator will be set.

You can determine if the zFS aggregate indicator is set by using IDCAMS LISTCAT ALL against the zFS aggregate and looking for the zFS indicator in the output.

Compatibility mode aggregates do not need to be separately attached because they are attached during MOUNT processing. However, if you want to issue a **`zfsadm`** command against a compatibility mode aggregate without mounting the aggregate,
| you can use the **`zfsadm attach`** command. You might attach an aggregate to grow
| it or display information about it.

Privilege required

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

Examples

The following command attaches an aggregate.

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

Figure 49 shows the same example as a job that invokes **`zfsadm attach`**.

```
//USERIDA  JOB ,'Zfsadm Attach',
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//AGGRINFO EXEC   PGM=IOEZADM,REGION=0M,
// PARM=('attach -aggregate OMVS.PRV.AGGR001.LDS0001')
//SYSPRINT DD     SYSOUT=H
//STDOUT   DD     SYSOUT=H
//STDERR   DD     SYSOUT=H
//SYSUDUMP DD     SYSOUT=H
//CEEDUMP  DD     SYSOUT=H
//*
```

Figure 49. Job to attach an aggregate

If you want to specify the `R/O` option, you must specify a leading slash. Otherwise, Language Environment® will treat the characters before the slash as Language Environment parameters. That is, you must use `PARM=('/attach OMVS.PRV.AGGR001.LDS0001 -R/O')`.

Related information

Commands:
- **`zfsadm fsinfo`** |
- **`zfsadm lsaggr`**

Files:
- IOEFSPRM

zfsadm config

Purpose

Changes the value of zFS configuration (IOEFSPRM) options in memory. See Chapter 12, "The zFS configuration options file (IOEPRMxx or IOEFSPRM)," on page 219 for a complete list of IOEFSPRM options.

Format

```
zfsadm config [-adm_threadsnumber] [-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] [-metaback_cache_size number[,fixed]]
              [-aggrgrow {on|off}] [-romount_recovery {on|off}]
              [-convert_auditfid {on|off}] [-client_reply_storage storage size]
              [-file_threads number] [-client_cache_size cache size[,fixed]]
              [-token_cache_size cache size] [-sysplex_filesys_sharemode {rwshare|norwshare}]
              [-change_aggrversion_on_mount {on | off}]
              [-format_aggrversion {4 | 5}] [-converttov5 {on | off}]
              [-system sysname] [-level] [-help]
|             [-modify_cmd_threads number]
```

Options

When you change options that apply to zFS aggregates and file systems, the current default changes. However, the change does not affect file systems that have already been mounted until they have been unmounted and remounted. Those options are as follows:

```
aggrfull
aggrgrow
convert_auditfid
change_aggrversion_on_mount
converttov5
fsfull
sysplex_filesys_sharemode
```

-adm_threads *number*
: Specifies the number of threads that are defined to handle pfsctl or mount requests.

-aggrfull *threshold,increment*
: Specifies the threshold and increment for reporting aggregate full error messages to the operator.

-aggrgrow on | off
: Specifies whether an aggregate should be dynamically extended when it runs out of physical space.

-change_aggrversion_on_mount on | off
: Specifies whether an aggregate should be changed to a version 1.5 aggregate on mount. The default is off.

-client_cache_size *cache size***[,fixed]**
: Specifies the size, in bytes, of the client cache. This is only meaningful
| when zFS is running sysplex-aware. This option is not supported; if it is
| specified, it is accepted but not used.

-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, or defaulted, mount (attach) changes the standard auditfid format to the unique auditfid format if the mount (attach) is read/write. If `off` is specified (or the mount (attach) is read-only), the auditfid is unaffected.

-converttov5 on | off
Specifies whether directories in a version 1.5 aggregate should be converted from v4 directories to extended (v5) directories as they are accessed. A version 1.4 aggregate is changed to a version 1.5 aggregate. You can override this setting at mount time by specifying CONVERTTOV5 or NOCONVERTTOV5.

-file_threads *number*
Specifies the current number of file threads. This option is only meaningful when zFS is running sysplex-aware.

-format_aggrversion 4 | 5
Specifies whether a version 1.4 aggregate or a version 1.5 aggregate should be formatted by default.

-fsfull *threshold,increment*
Specifies the threshold and increment for reporting file system full error messages to the operator.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

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

-log_cache_size *number* **[,fixed]**
Specifies the size, in bytes, of the cache that is 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 that is used to contain metadata. The fixed option reserves real storage for usage by zFS only.

| **-metaback_cache_size** *number*
| Specifies the size of the metadata backing cache. This size is combined
| with `meta_cache_size` to get the total size of the metadata cache.

| **-modify_cmd_threads** *number*
| Specifies the current number of threads that are defined to handle zFS
| modify commands.

-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 **<u>on</u> | off**
Controls the global new block security. zFS always runs with new block security on. The `off` option is not supported; if it is specified, it is accepted but not used.

-romount_recovery on | off
: Specifies whether zFS will automatically avoid a read-only mount failure (zFS reason code EFxx6271) because of the need to run log recovery for this aggregate. This can occur when the aggregate has been mounted read/write and a failure occurred before it was unmounted. If the next mount is for read-only, log recovery needs to run for the mount to be successful. If `on` is specified and this situation occurs, zFS temporarily mounts the aggregate read/write to allow log recovery to run and then zFS will unmount and then mount the aggregate read-only.

-sync_interval *number*
: Specifies the number of seconds between the times where zFS flushes data in its buffers to disk. The default is 30 seconds.

-sysplex_filesys_sharemode rwshare | norwshare
: Specifies the default for the mount PARM when a zFS read/write file system is mounted on a sysplex=filesys system. You can override this setting at mount time by specifying an alternate value in the actual mount PARM.

-system *sysname*
: Specifies the name of the system that the configuration option change request is sent to.

-token_cache_size *cache size*
: Specifies the token cache size maximum. When the token_cache_size is decreased, it is really the maximum size that is being decreased. This is only possible if the current usage is less than the maximum size. The token cache size cannot be decreased to lower than the current usage. The current usage is displayed through the MODIFY ZFS,QUERY,STKM command. This option 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. This option is
| not supported; if it is specified, it is accepted but not used.

-user_cache_size *number* **[,fixed]**
: Specifies the size, in bytes, of the cache that is 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 zFS will cache.

Usage

The **`zfsadm config`** command changes the configuration options (in memory) that were specified in the IOEFSPRM file (or defaulted). The IOEFSPRM file is not changed. If you want the configuration specification to be permanent, you must modify the IOEFSPRM file because zFS reads the IOEFSPRM file to determine the configuration values when zFS is started. The values that can be specified for each option are the same as the values that can be specified for that option in the IOEFSPRM file. You can specify that the configuration option change request should be sent to another system by using the `-system` option. The following options cannot be set using the **`zfsadm config`** command:
- `-cmd_trace`

- `-debug_dsn`
- `-group`
- `-msg_input_dsn`
- `-trace_table_size`
- `-sysplex_state`

Privilege required

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

Examples

The following example changes the size of the user cache:

```
zfsadm config -user_cache_size 64M

IOEZ00300I Successfully set -user_cache_size to 64M
```

Related information

Commands:
  `zfsadm configquery`

Files:
  IOEFSPRM

zfsadm configquery

Purpose

Queries the current value of zFS configuration options.

Format

```
zfsadm configquery [-system sysname] [-adm_threads] [-aggrfull] [-aggrgrow]
                   [-all] [-change_aggrversion_on_mount] [-client_cache_size] [-client_reply_storage]
                   [-cmd_trace] [-converttov5] [-convert_auditfid] [-debug_dsn] [-file_threads]
                   [-format_aggrversion]  [-fsfull] [-group] [-log_cache_size] [-meta_cache_size]
                   [-metaback_cache_size] [-msg_input_dsn] [-msg_output_dsn] [-nbs] [-romount_recovery]
                   [-sync_interval] [-syslevel] [-sysplex_filesys_sharemode] [-sysplex_state]
                   [-token_cache_size] [-trace_dsn] [-trace_table_size] [-tran_cache_size]
                   [-user_cache_size] [-vnode_cache_size] [-level] [-help]
|                  [-modify_cmd_threads]
```

Options

-adm_threads
Displays the number of threads that are defined to handle pfsctl or mount requests.

-aggrfull
Displays the threshold and increment for reporting aggregate full error messages to the operator.

-aggrgrow
Displays whether an aggregate should be dynamically extended when it runs out of physical space.

-all Displays the full set of configuration options.

-change_aggrversion_on_mount
Displays whether a version 1.4 aggregate should be changed to a version 1.5 aggregate when it is mounted.

-client_cache_size
Displays the size, in bytes, of the client cache. This is only meaningful
| when zFS is running sysplex-aware. If you use **`zfsadm config`** to set
| `-client_cache_size` to a value, the value is displayed but not used.

-client_reply_storage
Displays the number of bytes allocated for sysplex client reply storage. This is only meaningful when zFS is running sysplex-aware.

-cmd_trace
Displays whether command tracing is active.

-converttov5
Displays whether an aggregate should be assigned the converttov5 attribute on mount or remount. This attribute controls whether v4 directories will be converted to extended (v5) directories as they are accessed.

-convert_auditfid
Displays whether the zFS auditfid is automatically changed to the unique format on mount (attach). If `on` is specified or defaulted and the mount (attach) is read/write, the mount (attach) changes the standard auditfid format to the unique auditfid format. If `off` is specified or the mount (attach) is read-only, the auditfid is unaffected.

-debug_dsn
Displays the name of the debug input parameters data set.

-file_threads
Displays the current number of file threads. This option is only meaningful when zFS is running sysplex-aware.

-format_aggrversion
Displays whether an aggregate formatting default should be to format as a version 1.4 or 1.5 aggregate.

-fsfull
Displays the threshold and increment for reporting file system full error messages to the operator.

-group
Displays the XCF group that is used by zFS for communication between sysplex members.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

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

-log_cache_size
Displays the size, in bytes, of the cache that is used to contain buffers for log file pages.

-meta_cache_size
Displays the size, in bytes, of the cache that is used to contain metadata.

-metaback_cache_size
Displays the size of the backing cache for metadata.

| **-modify_cmd_threads**
| Displays the number of threads that are defined to handle zFS modify
| commands.

-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 zFS initialization output messages that come from the zFS PFS.

-nbs Controls the global new block security. zFS always runs with new block security on. If you use **`zfsadm config`** to set `-nbs` to off, it is displayed as off, but the value is not used.

-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 zFS 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, including:
- The version and release of z/OS

- The service level and FMID of zFS
- The date and time the PFS was built
- Whether the PFS is running sysplex-aware on a file system basis (referred to as filesys), or sysplex-aware on a system basis (referred to as file), or not sysplex-aware (referred to as admin-only), and the zFS XCF protocol level when running in a shared file system environment. (For information about the XCF protocol level, see "Determining the XCF protocol interface level" on page 97.) When filesys is indicated, the default mount PARM (NORWSHARE or RWSHARE) also displays.

This is the same information that is displayed by the operator command MODIFY ZFS,QUERY,LEVEL. In contrast, **`zfsadm configquery -level`** shows the level information for the **`zfsadm`** command itself.

-sysplex_filesys_sharemode
Displays the current default for the mount PARM (RWSHARE or NORWSHARE). It is only meaningful on systems that are running zFS sysplex=filesys.

-sysplex_state
Displays the sysplex state of zFS.

3 zFS is running in a sysplex-aware environment with sysplex=filesys.

-system *sysname*
Specifies the name of the system the report request is sent to retrieve the requested data.

-token_cache_size
Displays the current `token_cache_size` maximum. The current usage is displayed through the MODIFY ZFS,QUERY,STKM command. This option 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. If you use
| **`zfsadm config`** to set `-tran_cache_size` to a value, the value is displayed
| but not used.

-user_cache_size
Displays the size, in bytes, of the cache that is used to contain file data.

-vnode_cache_size
Displays the number of vnodes that will be cached by zFS.

Usage

The **`zfsadm configquery`** command displays the current value of zFS configuration options. The value is retrieved from zFS address space memory rather than from the IOEFSPRM file. You can specify that the configuration option query request should be sent to another system by using the `-system` option.

Ignore the following values when zFS is running non-sysplex aware. No storage is obtained even though a value might be reported.

- - client_cache_size
- - client_reply_storage
- - file_threads
- - token_cache_size

Privilege required

| If you are using an IOEFSPRM file in your startup proc, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

The following example displays the current value of the `user_cache_size` option:

```
zfsadm configquery -user_cache_size

IOEZ00317I The value for config option -user_cache_size is 64M.
```

To display all the zFS configuration options from each member, you can use the following command:

```
for sys in $(zfsadm lssys | grep -v IOEZ00361I); \
do; echo; echo $sys; zfsadm configquery -all -system $sys; done
```

Related information

Commands:
zfsadm config

Files:
IOEFSPRM

zfsadm convert

Purpose

Converts a v4 directory that is contained in a read/write mounted version 1.5 aggregate to an extended (v5) directory. The aggregate is changed from a version 1.4 aggregate to a version 1.5 aggregate, if necessary. It can also be used to change a version 1.4 aggregate to a version 1.5 aggregate without converting any directories.

Format

```
zfsadm convert {-path name | -aggrversion name} [-level] [-help]
```

Options

-aggrversion *name*
Specifies the aggregate name that should be changed from a version 1.4 aggregate to a version 1.5 aggregate. No directories are converted. The aggregate name is not case-sensitive. It is translated to uppercase.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

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

-path *name*
Specifies the path name of a directory that should be converted to an extended (v5) directory. The aggregate is changed to a version 1.5 aggregate first, if necessary.

Usage

The **zfsadm convert** command can be used to explicitly convert a v4 directory to an extended (v5) directory that is contained in a read/write mounted version 1.5 aggregate. In this case, the **-path** option is used. If the containing aggregate is a version 1.4 aggregate, the command attempts to change the aggregate to a version 1.5 aggregate before converting the directory.

It can also be used to explicitly change a version 1.4 aggregate to a version 1.5 aggregate without converting any directories. In this case, the **-aggrversion** option is used.

The **zfsadm convert** command might cause the file system to grow if it needs more space for the extended (v5) directory.

| The command must be issued from a z/OS V2R1 or later system and the zFS file
| system must be zFS-owned on a z/OS V2R1 or later system. The aggregate must be mounted read/write.

Do not use this command before you have migrated all your systems to z/OS
| V2R1 or later. If there are systems that are prior to z/OS V2R1 active in the shared file system environment, no conversion of a directory nor change of aggregate version takes place.

| If you use a job to invoke **zfsadm convert**, to specify the **-path** option, you must specify a leading slash in the PARM string if the path argument contains a slash. Otherwise, Language Environment will treat the characters before the slash as Language Environment parameters. That is, you must use `PARM=('/convert -path /home/myname/mydir')`.

Privilege required

| If you are using an IOEFSPRM file in your startup proc, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

The issuer must be the owner of the directory and must have write permission (`w`) to the directory. If the aggregate version is to be changed, the issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Examples

The following example contains the steps to convert an existing version 1.4 aggregate to a version 1.5 aggregate, and to convert a v4 directory to an extended (v5) directory.

1. To display the version of the aggregate:

```
# zfsadm aggrinfo PLEX.JMS.AGGR009.LDS0009 -long
PLEX.JMS.AGGR009.LDS0009 (R/W COMP): 1271 K free out of total 1440
version 1.4
auditfid C3C6C3F0 F0F200A2 0000

           158 free 8k blocks;          7 free 1K fragments
           112 K log file;             16 K filesystem table
             8 K bitmap file
```

2. To change the version to 1.5:

```
# zfsadm convert -aggrversion PLEX.JMS.AGGR009.LDS0009
IOEZ00810I Successfully changed aggregate PLEX.JMS.AGGR009.LDS0009 to version 1.5.
```

3. To verify the aggregate version change:

```
# zfsadm aggrinfo PLEX.JMS.AGGR009.LDS0009 -long
PLEX.JMS.AGGR009.LDS0009 (R/W COMP): 1271 K free out of total 1440
version 1.5
auditfid C3C6C3F0 F0F200A2 0000

           158 free 8k blocks;          7 free 1K fragments
           112 K log file;             16 K filesystem table
             8 K bitmap file
```

4. To display the version of a directory:

```
# zfsadm fileinfo /service9
   path: /service9
   ***   global data   ***
   fid                    1,1           anode                   69,516
   length                 8192          format                  BLOCKED
   1K blocks              8             permissions             755
   uid,gid                0,10          access acl              0,0
   dir model acl          0,0           file model acl          0,0
   user audit             F,F,F         auditor audit           N,N,N
   set sticky,uid,gid     0,0,0         seclabel                none
   object type            DIR           object linkcount        3
   object genvalue        0x00000000    dir version             4
   dir name count         3             dir data version        1
   dir tree status        VALID         dir conversion          na
   file format bits       na,na,na      file charset id         na
   file cver              na            charspec major,minor    na
   direct blocks          25
   indirect blocks        none
   mtime       Jun 13 15:27:10 2012   atime        Jun 13 10:41:43 2012
   ctime       Jun 13 15:27:10 2012   create time  Jun 13 10:41:43 2012
   reftime     none
```

5. To convert the directory to an extended (v5) directory:

```
# zfsadm convert -path /service
IOEZ00791I Successfully converted directory /service9 to version 5 format.
```

6. To display the version of the directory again:

```
# zfsadm fileinfo /service9
   path: /service9
   ***   global data   ***
   fid                    1,1           anode                   69,516
   length                 8192          format                  BLOCKED
   1K blocks              8             permissions             755
   uid,gid                0,10          access acl              0,0
   dir model acl          0,0           file model acl          0,0
   user audit             F,F,F         auditor audit           N,N,N
   set sticky,uid,gid     0,0,0         seclabel                none
   object type            DIR           object linkcount        3
   object genvalue        0x00000000    dir version             5
   dir name count         3             dir data version        1
   dir tree status        VALID         dir conversion          na
   file format bits       na,na,na      file charset id         na
   file cver              na            charspec major,minor    na
   direct blocks          25
   indirect blocks        none
   mtime       Jun 13 15:27:10 2012   atime        Jun 13 10:41:43 2012
   ctime       Jun 13 15:27:10 2012   create time  Jun 13 10:41:43 2012
   reftime     none
```

Related information

Commands:
   `zfsadm config`
| **`zfsadm fsinfo`**

Files:
   IOEFSPRM

zfsadm define

Purpose

Defines a VSAM linear data set that can be formatted as a zFS aggregate.

Format

```
zfsadm define -aggregate name  [-dataclass SMS_data_class]
                               [-managementclass SMS_management_class]
                               [-storageclass SMS_storage_class] [-catalog catalog]
                               [-system sysname] [-model model [catalog]]
                               [-volumes volume [volume ...]]
                               [-cylinders primary [secondary]] [-kilobytes primary [secondary]]
                               [-megabytes primary [secondary]] [-records 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 linear data set that is defined. The aggregate name is not case-sensitive. It is translated to uppercase.

-catalog *catalog*
: Specifies the name of the catalog in which the VSAM linear data set is to be defined.

-cylinders *primary* [*secondary*]
: Specifies the primary and optionally, the secondary allocation size for the VSAM linear data set in cylinders. The VSAM linear data set must have a secondary allocation size specified, if you want to use dynamic grow. See "Dynamically growing a compatibility mode aggregate" on page 30 for more information.

-dataclass *SMS_data_class*
: Specifies the name of the data class to be used when the VSAM linear data set is defined.

-help
: Prints the online help for this command. All other valid options specified with this option are ignored.

-kilobytes *primary* [*secondary*]
: Specifies the primary and optionally, the secondary allocation size for the VSAM linear data set in kilobytes. The VSAM linear data set must have a secondary allocation size specified, if you want to use dynamic grow. See "Dynamically growing a compatibility mode aggregate" on page 30 for additional information.

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

-managementclass *SMS_management_class*
: Specifies the name of the management class to be used when the VSAM linear data set is defined.

-megabytes *primary* [*secondary*]
: Specifies the primary and optionally, the secondary allocation size for the VSAM linear data set in megabytes. The VSAM linear data set must have a

secondary allocation size specified, if you want to use dynamic grow. See "Dynamically growing a compatibility mode aggregate" on page 30 for additional information.

-model *model* **[***catalog***]**
Specifies the name of the model and optionally, the model entry's catalog to be used when the VSAM linear data set is defined.

-records *primary* **[***secondary***]**
Specifies the primary and optionally, the secondary allocation size for the VSAM linear data set in records. When `records` is specified, the record size is assumed to be 4089 bytes. The VSAM linear data set must have a secondary allocation size specified, if you want to use dynamic grow. See "Dynamically growing a compatibility mode aggregate" on page 30 for additional information.

-storageclass *SMS_storage_class*
Specifies the name of the storage class to be used when the VSAM linear data set is defined.

-system *sysname*
Specifies the name of the system that the define request will be sent to.

-tracks *primary* **[***secondary***]**
Specifies the primary and optionally, the secondary allocation size for the VSAM linear data set in tracks. The VSAM linear data set must have a secondary allocation size specified, if you want to use dynamic grow. See "Dynamically growing a compatibility mode aggregate" on page 30 for additional information.

-volumes *volume*
Specifies the volume on which the VSAM linear data set can have space.

Usage

The **`zfsadm define`** command defines a VSAM linear data set. The VSAM linear data set is available to be formatted as a zFS aggregate. The command creates a DEFINE CLUSTER command string for a VSAM linear data set with SHAREOPTIONS(3) and passes it to the IDCAMS utility. If a failure occurs, the **`zfsadm define`** command can display additional messages from IDCAMS indicating the reason for the failure.

Privilege required

The issuer of the **`zfsadm define`** command requires sufficient authority to create the VSAM linear data set.

Examples

The following command defines a VSAM linear data set.

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

Related information

Commands:
`zfsadm format`

zfsadm delete

Purpose

Removes a backup file system in a compatibility mode aggregate. Beginning in z/OS V2R2, .bak file systems can only be deleted on aggregates that are zFS owned on downlevel systems.

This command will be removed in a future release.

Format

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

Options

-aggregate *name*
: Specifies the name of the aggregate where the zFS file system resides. It is specified to qualify the zFS file system name (`-filesystem`) when there are multiple zFS file systems with the same name in different aggregates. The aggregate name is not case-sensitive. It is always folded to uppercase.

-filesystem *name*
: Specifies the name of the backup file system to be removed. Include the .bak extension. The file system name is case-sensitive.

-help
: Prints the online help for this command. All other valid options that are specified with this option are ignored.

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

Usage

The **`zfsadm delete`** command removes the backup zFS file system that is indicated by the `-filesystem` option from its aggregate. The aggregate containing the file system to be deleted must be attached. Removing a backup file system does not remove the read/write file system.

Beginning in z/OS V2R2, no aggregates can be attached that contain more than one file system or a clone (.bak). Therefore, file systems can only be deleted from aggregates that are zFS owned on downlevel systems.

You can delete a compatibility mode file system (and its aggregate) by using the IDCAMS DELETE operation. This operation deletes the VSAM linear data set. For more information about renaming or deleting a compatibility mode aggregate, see "Renaming or deleting a compatibility mode aggregate" on page 39.

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 resource in the z/OS UNIXPRIV class. If you are instead using parmlib (IOEPRMxx), the issuer is required to be logged in as root or to have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Examples

The following command deletes the backup (clone) file system from its attached compatibility mode aggregate:

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

IOEZ00105I File System OMVS.USER.PAT.bak deleted successfully
```

Related information

Commands:
`zfsadm attach`
`zfsadm detach`
`zfsadm lsfs`

Files: File:
IOEFSPRM

zfsadm detach

Purpose

Detaches one or more aggregates from zFS. Any file systems contained in the detached aggregate are unavailable to zFS.

Format

```
zfsadm detach [{-aggregate aggregate name| -all [-system sysname]}] [-level] [-help]
```

Options

-aggregate *aggregate name*
: Specifies the aggregate name of the aggregate to be detached. Use this option or use `-all`, but not both. The aggregate name is not case-sensitive. It is always translated to uppercase.

-all
: Specifies that all attached aggregates in the sysplex are to be detached. Use this option or use `-aggregate` but not both.

-help
: Prints the online help for this command. All other valid options specified with this option are ignored.

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

-system *sysname*
: Specifies the name of the system where the aggregates to be detached reside. It cannot be specified without the `-all` option.

Usage

The **zfsadm detach** command is used to detach an aggregate. Detaching an aggregate makes it unavailable to the system. To detach one or more aggregates, use the `-all` or the `-aggregate` option to specify the aggregates to be detached. Use the `-system` option to limit the detach to a single system. The `-system` option cannot be specified without the `-all` option.

zfsadm detach does not detach mounted compatibility mode aggregates.

Privilege required

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

Examples

The following example shows a **zfsadm detach** command that detaches the aggregate OMVS.PRV.AGGR001.LDS0001.

```
zfsadm detach -aggregate omvs.prv.aggr001.lds0001

IOEZ00122I Aggregate OMVS.PRV.AGGR001.LDS0001 detached successfully
```

Related information

Commands:
`zfsadm attach`

Files:
IOEFSPRM

zfsadm fileinfo

Purpose

Displays detailed information about a file or directory.

Format

```
zfsadm fileinfo -path name [{-globalonly | -localonly | -both}] [-level] [-help]
```

Options

-both Causes the command to display both global and local information about the file or directory.

-globalonly
Causes the command to display global (on-disk) information about the file or directory. This option is the default.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

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

-localonly
Causes the command to display local (in memory on this system) information about the file or directory.

-path *name*
Specifies the path name of a file or directory about which information should be displayed. The path name is case-sensitive.

Usage

The **zfsadm fileinfo** command can be used to display information about a file or directory. It supports files and directories in version 1.4 aggregates. It also supports files and v4 or extended (v5) directories in version 1.5 aggregates.

If an aggregate has the `converttov5` attribute assigned to it, accessing a v4 directory with **zfsadm fileinfo** can cause its conversion to an extended (v5) directory. For more information, see “Converting an existing v4 directory to an extended (v5) directory” on page 28.

| The command must be issued from a z/OS V2R1 or later system. The file or directory must be contained in a file system that is locally zFS-owned or in a client file system.

| If you use a job to invoke **zfsadm fileinfo**, to specify the `-path` option you must
| specify a leading slash in the PARM string if the path argument contains a slash.
| Otherwise, Language Environment will treat the characters before the slash as
| Language Environment parameters. That is, you must use `PARM=('/fileinfo -path`
| `/home/myname/mydata')`.

Some of the fields are only applicable to files, some are only applicable to directories, some are only applicable to the local system and some are only applicable to client systems. There can also be attributes that are sometimes associated with a file or directory, such as ACLs. When these situations occur, the fields of the output display will contain values such as `0` or `na` or `none`, depending on the type of value that the field contains when it does have valid information.

If the `-globalonly` option is specified (or defaulted), the following fields are displayed:

fid The inode and uniquifier separated by a comma.

anode Anode block and offset into anode block separated by a comma.

length Length of data (directories are multiples of 8 K).

format
INLINE, FRAGMENTED, or BLOCKED.

1K blocks
Number of blocks that are used to store data, in kilobytes.

permissions
Permissions in octal format.

uid,gid
UID and GID of owner that is separated by a comma.

access acl
Anode index to ACL and length of ACL separated by a comma.

dir model acl
Anode index to directory model ACL and length of ACL separated by a comma.

file model acl
Anode index to file model ACL and length of ACL separated by a comma.

user audit
User audit flags for read, write, and execute:

N	None
S	Audit successful attempts
F	Audit failed attempts

auditor audit
Auditor audit flags for read, write, and execute:

N	None
S	Audit successful attempts
F	Audit failed attempts

set sticky,uid,gid
Sticky bit, set uid, and set gid separated by a comma.

seclabel
Security label for file or directory.

object type
DIR or FILE or LINK or CHARSPEC.

object linkcount
Link count for the object.

object genvalue
Object general attributes. This value is taken from at_genvalue in the z/OS UNIX structure ATTR.

dir version
The version of the directory; `5` indicates an extended (v5) directory and `4` indicates a v4 directory.

dir name count
The number of objects in an extended (v5) directory.

dir data version
A number that is incremented each time that the directory is changed.

dir tree status
For an extended (v5) directory, VALID (accessed by hash) or BROKEN (accessed as a flat file). Not applicable for a v4 directory.

dir conversion
For an extended (v5) directory, not applicable. For a v4 directory, FAILED (directory conversion was unsuccessful) or not applicable.

file format bits
For a file, the txt flag, the defer tag, the file format. For other objects, the text flag, the defer tag, and the file format are not applicable.

file charset id
The coded character set ID. This value is taken from at_charsetid in the z/OS UNIX structure ATTR.

file cver
Creation verifier. This value is taken from AT_cver in the z/OS UNIX structure ATTR.

charspec major,minor
Character special file, major number, minor number. Each character special file has a device major number, which identifies the device type, and a device minor number, which identifies a specific device of a given device type.

direct blocks
The block numbers of the first eight 8-K blocks.

indirect blocks
The block numbers of the level 0, level 1 and level 2 trees.

mtime Last modification time.

atime Last access time.

ctime Last change time.

create time
Create time.

reftime
Last reference time.

If the `-localonly` option is specified, the following fields are displayed:

vnode,vntok
Addresses of the ZFS vnode and the z/OS UNIX vnode.

opens

ow	Number of tasks that are waiting to open due to deny mode opens
oi	Number of internal opens
rd	Number of read opens
rw	Number of write opens

open deny

rd	Number of deny-read opens
wr	Number of deny-write opens
ar	Number of advisory deny-read opens
aw	Number of advisory deny-write opens

owner zFS owning system.

file seq read
Indicates whether user file cache considers file to be read sequentially. Valid only for files.

file seq write
Indicates whether user file cache considers file to be written sequentially. Valid only for files.

file unscheduled
Indicates the number of unscheduled pages (dirty data) in the user file cache for files.

file segments
The number of 64 K segments of the file that is cached in the user file cache.

file dirty segments
The number of dirty segments in the user file cache. *Dirty segments* are regions of the file that are either dirty and not yet written to disk, or are waiting for an I/O to disk to complete.

file meta issued
Applicable to files or directories that were accessed by the sysplex client. It indicates whether the client made a request recently to the server where the object's metadata was updated.

file meta pending
Applicable to files or directories that are accessed by sysplex client. It indicates whether the client has an outstanding request to the server where the object's metadata might be updated.

client cached fsp
Indicates that the client has security information that is cached for the directory or file.

client cached anode
Indicates that the client has the object's attributes and location information for the directory or file.

client cached symlink
Indicates that the content of a symbolic link was cached by the sysplex client. This flag is valid only for symbolic links.

client revoke
Indicates whether a revoke is in progress to this sysplex client for this file or directory.

client thrashing
Indicates whether the file or directory is considered thrashing by zFS, and as a result, uses the zFS thrash resolution interface to the server.

client token rights
Indicates the token rights that are held by the sysplex client for the object.

client thrash ops
: Number of forwarded requests.

client ops to server
: Number of requests that the client made to the server for this object.

client meta buffers
: Number of buffers in the metadata or backing cache for this object for the sysplex client.

client meta updates
: Indicates whether the sysplex client has updated metadata for this object.

dirty meta buffers
: For owners, indicates the number of dirty buffers in the metadata cache for this file or directory.

Privilege required

If you are using an IOEFSPRM file in your startup proc, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

The issuer must have lookup authority (x) to the directory and READ authority (r) to the file.

Examples

The following example displays information for the `/service9` directory:

```
zfsadm fileinfo -both /service9
   path: /service9
   ***   global data   ***
   fid                  1,1          anode                 69,516
   length               8192         format                BLOCKED
   1K blocks            8            permissions           755
   uid,gid              0,10         access acl            0,0
   dir model acl        0,0          file model acl        0,0
   user audit           F,F,F        auditor audit         N,N,N
   set sticky,uid,gid   0,0,0        seclabel              none
   object type          DIR          object linkcount      2
   object genvalue      0x00000000   dir version           4
   dir name count       na           dir data version      0
   dir tree status      na           dir conversion        na
   file format bits     na,na,na     file charset id       na
   file cver            na           charspec major,minor  na
   direct blocks        107
   indirect blocks      none
   mtime       Jun 13 10:41:43 2012   atime        Jun 13 10:41:43 2012
   ctime       Jun 13 10:41:43 2012   create time  Jun 13 10:41:43 2012
   reftime     none
   ***   local data from system DCEIMGVM   ***
   vnode,vntok          0x00000000,,0x794C0900      0x00FF7CA0,,0x00000000
   opens                ow=0          oi=0          rd=0          wr=0
   open deny            rd=0          wr=0          ar=0          aw=0
   owner                DCEIMGVM   file seq read          na
   file seq write       na         file unscheduled       na
   file pending         na         file segments          na
   file dirty segments  na         file meta issued       na
   file meta pending    na         client cached fsp      na
   client cached anode  na         client cached symlink  na
   client revoke        na         client thrashing       na
   client token rights  na         client thrash ops       na
   client ops to server na         client meta buffers     na
   client meta updates  na         dirty meta buffers      0
```

Related information

Commands:

| **zfsadm fsinfo**

zfsadm format

Purpose

| Formats a VSAM linear data set to become a version 4 or version 5 zFS compatibility mode aggregate.

Format

```
zfsadm format -aggregate name [-initialempty blocks] [-size blocks] [-logsize blocks]
                              [-owner {uid | name}] [-group {gid | name}]
                              [-perms decimal | octal | hex_number] [-grow blocks]
                              [-system sysname] [-compat] [-overwrite]
                              [{-newauditfid] | -nonewauditfid}] [{-version4 | -version5}]
                              [-level] [-help]
```

Options

-aggregate *name*
: Specifies the name of the aggregate to be formatted. The aggregate name is not case-sensitive. It is translated to uppercase.

-compat
: Specifies that the zFS aggregate should be formatted as a compatibility mode aggregate. That is, it should be formatted as an aggregate and then a zFS file system should be created in the aggregate. The zFS file system will have the same name as the aggregate. `-compat` is the default but is ignored.

-group {*gid* | *name*}
: Specifies the group owner of the root directory of the file system. It can be specified as a z/OS group ID or as a GID. The default is the GID of the issuer of the **zfsadm format** command. If only `-owner` is specified, the group is that owner's default group.

-grow *blocks*
: Specifies the number of 8 KB blocks that zFS uses as the increment for extension when the `-size` option specifies a size greater than the primary allocation.

-help
: Prints the online help for this command. All other valid options that are specified with this option are ignored.

-initialempty *blocks*
: This option is being allowed for compatibility with earlier versions and is ignored. One 8-KB block at the beginning of the aggregate is reserved for IBM use.

-level
: Prints the level of the **zfsadm** command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

-logsize *blocks*
: Specifies the size in 8 KB blocks of the log. The valid range is from 13 to 16384 blocks (128 megabytes). The default is 1% of the aggregate size. This default logsize will never be smaller than 14 blocks and it will never be larger than 4096 blocks (32 megabytes). This size is normally sufficient. However, a small aggregate that is grown to be very large will still have a small log. You might want to specify a larger log if you expect the aggregate to grow very large.

-newauditfid
: Specifies that the aggregate should be formatted with the zFS auditfid and stored in the aggregate. This is the default.

-nonewauditfid
: Specifies that the aggregate should not be formatted with a zFS auditfid stored in it.

-overwrite
: Specifies that an existing zFS aggregate should be overlaid. All existing data is lost. Use this option with caution. This option is not usually specified.

-owner {*uid* | *name*}
: Specifies the owner of the root directory of the file system. It can be specified as a z/OS user ID or as a UID. The default is the UID of the issuer of the **`zfsadm format`** command.

-perms *number*
: Specifies the permissions of the root directory of the file system. It can be specified as an octal number (for example, o755), as a hexadecimal number (for example, x1ED), or as a decimal number (for example, 493). The default is o755 (owner read/write-execute, group read-execute, and other read-execute.)

-size *blocks*
: Specifies the number of 8 KB blocks that should be formatted to form the zFS aggregate. The default is the number of blocks that fits in the primary allocation of the VSAM linear data set. If a number less than the default is specified, it is rounded up to the default. If a number greater than the default is specified, a single extend of the VSAM linear data set is attempted after the primary allocation is formatted unless the `-grow` option is specified. In that case, multiple extensions of the amount that is specified in the `-grow` option are attempted until the `-size` is satisfied. Space must be available on the volume.

-system *sysname*
: Specifies the system that the format request will be sent to.

-version4
: Specifies that the aggregate should be a version 1.4 aggregate. See "Usage notes for zfsadm format" for the default value that is used.

-version5
: Specifies that the aggregate should be a version 1.5 aggregate. See "Usage notes for zfsadm format" for the default value that is used.

Usage notes for zfsadm format

The **`zfsadm format`** command formats a VSAM linear data set as a zFS aggregate. All zFS aggregates must be formatted before use. The **`zfsadm format`** command requires the zFS PFS to be active on the system. The size of the aggregate is as many 8-KB blocks as fits in the primary allocation of the VSAM linear data set or as specified in the `-size` option. To extend it, use the **`zfsadm grow`** command. If `-overwrite` is specified, all existing primary and secondary allocations are formatted and the size includes all of that space, and the backup change activity flag is set.

| If the VSAM linear data set has a SHAREOPTIONS value of other than 3, **`zfsadm`**
| **`format`** changes it to SHAREOPTIONS 3 during format.

| If the `-overwrite` option is specified, the backup change flag is set.

The aggregate version will be as specified if the `-version4` or `-version5` argument is used. If neither is used, then the default aggregate version will be obtained from the zFS PFS `format_aggrversion` setting. See "IOEFSPRM" on page 220 for a description of the `format_aggrversion` variable.

Privilege required

The issuer of the **`zfsadm format`** command must meet one of the following authorization requirements:

- Have ALTER authority to the VSAM linear data set.
- Be UID 0.
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

| If you are using an IOEFSPRM file in your startup proc, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

The following command formats the VSAM linear data set as a compatibility mode aggregate.

```
zfsadm format -aggregate omvs.prev.aggr001.lds0001 -owner usera -group audit -perms o750
```

Related information

Commands:
`zfsadm define`

Files:
IOEFSPRM

zfsadm fsinfo

Purpose

Displays detailed information about a zFS file system, which is also known as a *zFS aggregate*.

Format

```
zfsadm fsinfo
[-aggregate name | -path path | -all]
[{-basic |-owner | -full |-reset}]
[-select criteria | -exceptions]
[-sort sort_name][-level][-help]
```

Options

-aggregate *name*
: Specifies the name of the aggregate to be displayed. The aggregate name is not case-sensitive and is translated to uppercase. To specify multiple aggregates with similar names, use an asterisk (*) at the beginning, at the end, or both at the beginning and the end of *name* as a wildcard. If **-aggregate** *name* is specified with wildcards, the default display is **-basic**. Otherwise, the default display is **-owner**. See "Usage notes for zfsadm fsinfo" on page 192 for more information.

-all
: Displays information for all aggregates in the sysplex. It is the default when **-aggregate** and **-path** are not specified. The default information display will be as if **-basic** were specified.

-basic
: Displays a line of basic file system information for each specified file system. This option is the default in the following situations:

- The **-all** option is specified but **-full**, **-owner**, and **-reset** are not specified.
- None of **-aggregate**, **-all**, **-path**, **-full**, **-owner**, and **-reset** options are specified.
- The **-sort** and **-exceptions** options are specified and neither **-full** nor **-owner** is specified.
- The **-aggregate** option is specified with one or more wildcards.

See "Usage notes for zfsadm fsinfo" on page 192 for more information.

-exceptions
: Displays information about any specified aggregate that is quiesced, disabled, had grow failures, is low on space or damaged. Any specified aggregate is also displayed if it has had XCF communication failures or an error because it ran out of space or when doing an I/O operation. This option cannot be specified with **-reset**, **-path**, **-select** and **-aggregate** with no wildcard in *name*. Information is displayed by default as if the **-basic** option were specified. See "Usage notes for zfsadm fsinfo" on page 192 for more information.

-full
: Displays information that is maintained by the system owning each specified file system. See Table 14 on page 194 for a description of the information displayed for the owner. It also displays information that is locally maintained by each system in the sysplex that has each specified

file system locally mounted. For information about local statistics that are displayed when the **-full** option is specified, see Table 16 on page 196.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

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

-owner
Displays only information that is maintained by the system owning each specified file system. This option is the default when **-aggregate** without wildcards is specified. See "Usage notes for zfsadm fsinfo" for more information.

-path *path*
Specifies the path name of a file or directory that is contained in the file system for which information is to be displayed. The path name is case-sensitive and can start with or without a slash (/). The default information display will be as if **-owner** were specified.

-reset Resets zFS statistics that are related to each specified file system.

-select *criteria*
Displays each specified file system that matches the criteria. Information is displayed by default as if the **-basic** option were specified. The information that is displayed can also be sorted by using the **-sort** option.

To use this option, specify a selection criteria from Table 12 on page 193.

This option cannot be specified with **-exceptions**, **-reset**, **-path**, and **-aggregate** with no wildcard in *name*. See "Usage notes for zfsadm fsinfo" for more information.

-sort *sort_name*
Specifies that the information displayed is to be sorted as specified by the value of *sort_name*. The default is sort by Name. This option cannot be specified with **-reset**. The valid sorting options are listed in Table 15 on page 196.

Usage notes for zfsadm fsinfo

1. The **zfsadm fsinfo** command displays detailed information about the specified file systems. Normally, file systems must be attached before this command can be used to display their information. However, when a specific aggregate name (with no wildcards) is specified, the file system does not need to be attached. You can use several methods to specify aggregates, based on their names, as follows:
   - **-aggregate** with an exact aggregate name. The aggregate name is not case-sensitive and is translated to uppercase.
   - **-aggregate** using a wildcard ('*') at the beginning of the name value to select aggregates with a common suffix.
   - **-aggregate** using a wildcard ('*') at the end of the name value to select aggregates with a common prefix.
   - **-aggregate** using a wildcard ('*') at the beginning and the end of the name value to select aggregates with both a common prefix and a common suffix.
   - **-path** with the path name of a file or directory in a zFS file system. Information for the file system that contains the file or directory is displayed.

Note: To ensure proper processing by the z/OS UNIX shell, put single quotation marks around the wildcard (*).

The **-all** option selects all file systems that are attached in the sysplex. It is the default.

2. The **-owner** option displays all available information for each specified file system from the zFS-owning system. The information is obtained via XCF communication with the owning system if the owning system is not the local system.
3. Aggregates can be selected by use of the **-select** option. To use this option, specify a criteria from the following table. You can specify more than one criteria by using a comma to separate them. Table 12 lists the criteria that you can use.

Table 12. Criteria for selecting aggregates

Criteria	Shows aggregates that ...
CE	Had XCF communication failures between client systems and owning systems. This typically means that applications have gotten timeout errors.
DA	Are marked damaged by the zFS salvager.
DI	Are disabled for reading and writing.
GD	Are disabled for dynamic grow.
GF	Had failed dynamic grow attempts.
GR	Are currently being grown.
IE	Have had disk I/0 errors.
L	Have less than 1 MB of free space, which means that increased XCF traffic is required for writing files.
NS	Are mounted NORWSHARE.
OV	Contain extended (v5) directories that are using overflow pages.
Q	Are currently quiesced.
RO	Are mounted read-only.
RQ	Had application activity.
RW	Are mounted read/write.
RS	Are mounted RWSHARE.
SE	Have returned ENOSPC errors to applications.
TH	Have sysplex thrashing objects in them.
V4	Are version 1.4.
V5	Are version 1.5.
V5D	Are disabled for conversion to version 1.5.
WR	Had application write activity.

4. Aggregates can be selected using the **-exceptions** option. This option can be useful for identifying file systems that have encountered unexpected conditions, and might need attention. Unexpected conditions include I/O errors, XCF communication failures or being low on space. An aggregate can also be damaged, quiesced, or disabled.
5. The **-basic** option displays the file system name, the zFS-owning system name, and file system status. Table 13 on page 194 lists the values of the file system

status. A Legend string is also displayed at the end of the output as a quick reference to show the definitions of the abbreviated status values.

6. When you use the **-owner** option, the displayed information has the file system status as part of the output. The status field contains abbreviated values. For quick reference, these values are defined in a Legend string at the end of the output. The full definitions of these abbreviations are listed in Table 13.

Table 13. Definitions of abbreviated values when the -basic or -owner options are specified

Values	Definition
CE	The aggregate had XCF communication failures (timeout errors) since the last statistics reset.
DA	Salvage considered the aggregate damaged and it has not been repaired yet.
DI	The aggregate is disabled for access.
GD	Dynamic grow was disabled. This is set if an aggregate has the AGGRGROW attribute assigned to it but due to a dynamic grow failure will not attempt future dynamic grows until an explicit administrator grow command is issued against that file system.
GF	Had failed dynamic grow attempts.
GR	The aggregate is being grown.
IE	The aggregate had disk I/O errors since the last statistics reset.
L	The aggregate is low on space as defined by the zFS distributed bitmap reservation algorithms (less than 1 MB of free space left).
NM	The aggregate is attached, but not mounted.
NS	The aggregate is mounted NORWSHARE, or the aggregate is attached.
OV	The aggregate has directories with overflow pages.
Q	The aggregate is quiesced.
RO	The aggregate is mounted in R/O mode.
RQ	Had application activity.
RW	The aggregate is mounted R/W.
RS	The aggregate is mounted RWSHARE.
SE	The aggregate ran out of space at some time since the last statistics reset.
TH	The aggregate has sysplex thrashing objects.

7. The **-owner** option displays the statistics that are shown in Table 14.

Table 14. Statistics displayed when the -owner option is specified

Statistics	Description
Owner	The name of the system that currently owns the aggregate.
Status	The status of the aggregate as known by the owning system. The display is a subset of the information that is available in the **-basic** display because it shows only what the owner knows. The **-basic** display is a one-line summary for all chosen sysplex members.

Table 14. Statistics displayed when the -owner option is specified (continued)

Statistics	Description
Converttov5	Indicates whether the file system has the CONVERTTOV5 attribute assigned to it. If the aggregate is version 1.4, or is version 1.5 and does not have the CONVERTTOV5 attribute assigned to it, the second value will be `n/a`. If the aggregate has the CONVERTTOV5 attribute assigned to it, the second value will indicate whether automatic conversion is ENABLED or DISABLED. One possible reason it could be DISABLED is that the aggregate was quiesced after this system assumed ownership of the file system.
Size	Size of the aggregate in kilobytes.
Free 8K Blocks	Number of free 8 K blocks.
Free 1K Fragments	Number of free fragments in partially allocated blocks.
Log File Size	Total space in kilobytes occupied by the log file, including indirect blocks.
Bitmap Size	Size of the bitmap file in kilobytes, including indirect blocks.
Anode Table Size	Total space occupied by the anode table in kilobytes, including indirect blocks.
File System Objects	The number of objects in the file system. This includes files, directories, symbolic links, ACLs and z/OS UNIX special files.
Version	The version of the aggregate. For example, 1.4 or 1.5.
Overflow Pages	The number of overflow pages that are allocated to extended (v5) directories.
Overflow HighWater	The highest number of overflow pages that were ever allocated on disk in extended (v5) directories.
Thrashing Objects	The current number of sysplex thrashing objects in the file system at one time.
Thrash Resolutions	The number of times the owner invoked the thrash resolution protocol (as opposed to the normal direct I/O protocol) to resolve sysplex contention of objects in the file system.
Token Revocations	The number of times the owner revoked tokens from other sysplex members, which means there was contention on an object and a callback had to be made to one or more clients.
Revocation Wait Time	The average time that it took to revoke tokens from clients.
Space Monitoring	The threshold and increment for space monitoring. `0,0` is used to mean that there is no space monitoring in use for the file system.
Devno	z/OS UNIX device number for the mounted file system.
Quiesce Jobname	Name of job that quiesced the aggregate.
Quiesce System	Name of the system where the application was running that quiesced the aggregate.
Quiesce ASID	ASID of the job that quiesced the aggregate.
File System Grow	Shows whether the Aggrgrow attribute is enabled (ON or OFF). It also shows the number of grows that were performed since this system assumed ownership of the file system.
Audit Fid	The audit fid that is used to represent the file system for SAF auditing.
File System Creation Time	Time that the file system was last formatted.
Time of Ownership	Time that the current owning system assumed ownership of the file system. That is, the time of its primary mount or when it last assumed ownership due to aggregate movement.
Statistics Reset Time	Time that the owner statistics were last reset.
Quiesce Time	The time that the file system was last quiesced.
Last Grow Time	The time that the file system was last grown (by command or dynamically) since this system assumed ownership of the file system.

| *Table 14. Statistics displayed when the -owner option is specified (continued)*

Statistics	Description
Connected Clients	This lists all client systems in the sysplex that have local mounts for a file system that is mounted RWSHARE.

8. Table 15 lists the sorting options when the **-sort** option is specified.

Table 15. Sorting options when the -sort option is specified

Sorting option	Function
Name	Sort by file system name, in ascending order. This sorting option is the default.
Requests	Sort by the number of external requests that are made to the file system by user applications, in descending order. The most actively requested file systems are listed first.
Response	Sort by response time of requests to the file system, in descending order. The slower responding file systems are listed first.

9. The **-full** option displays statistics for each specified file system from the zFS owning system and from each system in the sysplex that has it locally mounted. This is obtained via XCF communication with each system in the sysplex. The owning system statistics are described in Table 14 on page 194. The local statistics are described in Table 16.

| *Table 16. Local statistics displayed when the full option is specified*

Statistics	Description
| **Application Reads**	The number of read requests that were made by applications for files and directories in this file system.
| **Application Writes**	The number of write requests that were made by applications for files or directories in this file system.
| **Average**	The average task wait time when it had to wait for an I/O operation. This is the full wait time, including any queue wait time and device response time.
Avg. Rd XCF Resp. Time	The average response time for XCF read requests for objects on the owning system.
| **Avg. Read Resp. Time**	The average response time for read requests that were made by applications for files or directories in this file system.
Avg. Wr XCF Resp. Time	The average response time for XCF write requests for objects on the owning system.
| **Avg. Write Resp. Time**	The average response time for write requests that were made by applications for files or directories in this file system.
Canceled Operations	The number of times a task was asynchronously abended (forced or canceled) while accessing this file system.
DDNAME	The DDNAME for the data set allocation on this system.
Disk IO Errors	The number of disk I/O errors for disk I/O operations performed on this system.
| **ENOSPC Errors**	The number of out of space (ENOSPC) errors that were seen by applications for this file system on this system.
Kbytes	The number of kilobytes read from the DASD volume for this system.
LFS Held Vnodes	The number of vnodes that the z/OS UNIX logical file system has allocated for the file system.
Metadata Cache 8K Pages	The number of 8 K pages in the metadata cache for this file system.
Mount Time	The time the file system was mounted on this system.

Table 16. Local statistics displayed when the full option is specified (continued)

Statistics	Description
Open objects	Number of files or directories that are open.
PAV	The number of noncritical concurrent I/O operations that zFS will send to the DASD at one time for this DASD volume. zFS will send up to the device PAV capability I/O operations in parallel for critical I/O operations. An I/O operation is deemed critical if a task is, or will be waiting on that I/O operation to complete.
Quiesce Waiters	YES if there are tasks that are waiting for the file system to be unquiesced. Otherwise, NO.
Reads	The number of disk reads to the DASD volume for this system.
Read XCF Calls	The number of XCF requests to read objects from the system that owns the file system. This will be zero (0) on the owning system.
Statistics Reset Time	The time that the statistics for the local file system were last reset.
Tokens	The number of tokens that are held for objects in the file system by the token manager.
TOTALS	The totals for all DASD volumes for the file system on this system.
User Cache 4K Pages	The number of 4 K pages in the user file cache for this file system.
Vnodes	Number of vnodes in memory for the file system.
VOLSER	The DASD VOLSER that the file system resides on.
Waits	The number of times a task had to wait for an I/O operation to complete for disk I/O operations on this system.
Writes	The number of disk writes to the DASD volume for this system.
Write XCF Calls	The number of XCF requests to write objects to the system that owns the file system. This will be zero (0) on the owning system.
XCF Comm. Failures	The number of XCF communication failures (for example, timeouts) on XCF requests made for this file system on this system.

10. All times are in milliseconds. Large numbers are displayed using the following suffixes:

t	Multiply the shown value by 1,000,000,000.
m	Multiply the shown value by 1000000.
t	Multiply the shown value by 1000.
tr	Multiply the shown value by 1,000,000,000,000.
K	Multiply the shown value by 1024.
M	Multiply the shown value by 1048576.

Privilege required

If you are using an IOEFSPRM file in your startup 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.

To use the **-reset** option, the user must meet one of the authorization requirements:
- Be UID 0
- Have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class

Examples

1. To display basic file system information for zFS aggregate PLEX.DCEIMGNK.FSINFO:

```
zfsadm fsinfo -aggregate PLEX.DCEIMGNK.FSINFO -basic
PLEX.DCEIMGNK.FSINFO                        DCEIMGNJ  RW,RS,Q,GF,GD,L,SE
Legend: RW=Read-write, Q=Quiesced, GF=Grow failed, GD=Grow disabled
       L=Low on space, RS=RWSHARE mounted, SE=Space errors reported
```

2. To display full file system status for zFS aggregate PLEX.DCEIMGNK.FSINFO:

```
zfsadm fsinfo -aggregate PLEX.DCEIMGNK.FSINFO -full
File System Name: PLEX.DCEIMGNK.FSINFO

  *** owner information ***
  Owner:              DCEIMGNJ        Converttov5:            ON,DISABLED
  Size:               336K            Free 8K Blocks:         23
  Free 1K Fragments:  0               Log File Size:          112K
  Bitmap Size:        8K              Anode Table Size:       8K
  File System Objects: 3              Version:                1.5
  Overflow Pages:     0               Overflow HighWater:     0
  Thrashing Objects:  0               Thrashing Resolution:   0
  Token Revocations:  0               Revocation Wait Time:   0
  Devno:              46              Space Monitoring:       0,0
  Quiescing System:   DCEIMGNJ        Quiescing Job Name:     SUIMGNJ
  Quiescor ASID:      x4c             File System Grow:       ON,0
  Status:             RW,RS,Q,GF,GD,L,SE
  Audit Fid:          00000000        00000000                 0000

  File System Creation Time: Nov  5 15:15:54 2013
  Time of Ownership:         Nov  5 15:25:32 2013
  Statistics Reset Time:     Nov  5 15:25:32 2013
  Quiesce Time:              Nov  5 15:28:39 2013
  Last Grow Time:            n/a

  Connected Clients:  DCEIMGNK


Legend: RW=Read-write, Q=Quiesced, GF=Grow failed, GD=Grow disabled
       L=Low on space, RS=RWSHARE mounted, SE=Space errors reported

  *** local data from system DCEIMGNJ (owner: DCEIMGNJ) ***
  Vnodes:             1               LFS Held Vnodes:         4
  Open Objects:       0               Tokens:                  3
  User Cache 4K Pages: 5              Metadata Cache 8K Pages: 6
  Application Reads:  167837          Avg. Read Resp. Time:    0.059
  Application Writes: 23460           Avg. Writes Resp. Time:  0.682
  Read XCF Calls:     0               Avg. Rd XCF Resp. Time:  0.000
  Write XCF Calls:    0               Avg. Wr XCF Resp. Time:  0.000
  ENOSPC Errors:      0               Disk IO Errors:          0
  XCF Comm. Failures: 0               Cancelled Operations:    0

  DDNAME:             SYS00004
  Mount Time:         Nov  6 09:46:44 2013

VOLSER PAV    Reads      KBytes     Writes     KBytes     Waits    Average
______ ___ __________ __________ __________ __________ __________ _________
CFC001   1         12         88      25767     304116      18796 1.032
______ ___ __________ __________ __________ __________ __________ _________
TOTALS             12         88      25767     304116      18796 1.032
```

3. To display the status of the file system owner using a wildcard:

```
zfsadm fsinfo -aggregate PLEX.DCEIMGNJ.FS'*'

PLEX.DCEIMGNJ.FS1                            DCEIMGNJ RW,NS
PLEX.DCEIMGNJ.FS2                            DCEIMGNJ RW,RS
PLEX.DCEIMGNJ.FS3                            DCEIMGNJ RW,NS
PLEX.DCEIMGNJ.FS2                            DCEIMGNJ RW,RS
PLEX.DCEIMGNJ.FS3                            DCEIMGNJ RW,NS
Legend: RW=Read-write,NS=Mounted NORWSHARE,RS=Mounted RWSHARE
```

4. A job to obtain the file system information, using a wildcard:

```
//USERIDA JOB ,'Zfsadm fsinfo',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//GETINFO EXEC PGM=IOEZADM,REGION=0M,
// PARM=('fsinfo -aggregate PLEX.DCEIMGNJ.FS*')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
```

The following lines are possible output from the job:

```
PLEX.DCEIMGNJ.FS1                          DCEIMGNJ RW,NS
Legend: RW=Read-write,NS=Mounted NORWSHARE
```

5. A job to obtain information for the file system that contains directory `/u/userida/fs1`:

```
//USERIDA JOB ,'Zfsadm fsinfo',
// CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//GETINFO EXEC PGM=IOEZADM,REGION=0M,
// PARM=('/fsinfo -path /u/userida/fs1')
//SYSPRINT DD SYSOUT=H
//STDOUT DD SYSOUT=H
//STDERR DD SYSOUT=H
//SYSUDUMP DD SYSOUT=H
//CEEDUMP DD SYSOUT=H
```

The following lines are possible output from the job:

```
PLEX.DCEIMGNJ.FS1                          DCEIMGNJ RW,NS
Legend: RW=Read-write,NS=Mounted NORWSHARE,RS=Mounted RWSHARE
```

Related information

Commands:
   zfsadm aggrinfo
   zfsadm lsaggr
   zfsadm lsfs

Files:
   IOEFSPRM
   MODIFY ZFS PROCESS

zfsadm grow

Purpose

Makes the physical size of an aggregate larger.

Format

```
zfsadm grow -aggregate name -size kbytes [-level] [-help]
```

Options

-aggregate *name*
Specifies the name of the aggregate to be grown. The aggregate name is not case-sensitive. It is always translated to uppercase.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

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

-size *kbytes*
Specifies the new total size in kilobytes of the aggregate after the grow operation. The size is rounded up to a control area (CA). A control area is normally a cylinder or less and is based on the primary and secondary allocation units. See *z/OS DFSMS Using Data Sets* for more information about allocation size boundary. If zero is specified, the secondary allocation size is used. The value that is specified cannot exceed the size of a single volume.

Usage

The **zfsadm grow** command attempts to extend the size of an aggregate when the size specified is greater than the current size of the aggregate or when the size is specified as zero. If the extend fails (for example, if there is no space on the volume, or if size zero is specified and there is no secondary allocation specified for the VSAM linear data set), the grow operation fails. If the size specified is less than or equal to the current size of the aggregate, no extend is attempted and the command successfully returns. An aggregate cannot be made smaller than its current size. In any case, if the aggregate's high used value is less than the aggregate's high allocated value, the aggregate will be formatted up to the high allocated value (making the high used value equal to the high allocated value). The current (formatted) size of an aggregate can be determined by using the **zfsadm aggrinfo** command. The high used value (HI-U-RBA) and the high allocated value (HI-A-RBA) can be determined by using the IDCAMS LISTCAT ALL command. For an explanation of the rules that apply to extending a VSAM linear data set, see *z/OS DFSMS Using Data Sets*.

The size of the file system free space is increased by the amount of additional space available.

Privilege required

The issuer must have READ authority to the data set that contains the IOEFSPRM file and must be logged in as root or have READ authority to the

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

Examples

The following command displays the online help entry for the **zfsadm grow** command:

```
zfsadm grow -help

Usage: zfsadm grow -aggregate <name> -size <size in K bytes> [-level] [-help]
```

Related information

Command:
  zfsadm aggrinfo
  zfsadm fsinfo

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 that are specified with this option are ignored.

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

-topic *command*
Specifies each command whose syntax is to be displayed. Provide only the second part of the command name (for example, **lsfs**, not **zfsadm lsfs**). Multiple topic strings can be specified. If this option is omitted, the output provides a short description of all **zfsadm** commands.

Usage

The **zfsadm help** command displays the first line (name and short description) of the online help entry for every **zfsadm** command if `-topic` is not provided. For each command name specified with `-topic`, the output lists the entire help entry.

The online help entry for each **zfsadm** command consists of the following two lines:
- The first line names the command and briefly describes its function.
- The second line, which begins with `Usage:`, lists the command options in the prescribed order.

Use the **zfsadm apropos** command to show each help entry containing a specified string.

Privilege required

| If you are using an IOEFSPRM file in your startup proc, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

The following command displays the online help entry for the **zfsadm lsfs** command and the **zfsadm lsaggr** command:

zfsadm help -topic lsfs lsaggr

```
zfsadm lsfs: list filesystem information
Usage: zfsadm lsfs [-aggregate <aggregate name>] [{-fast | -long}] [-level] [-help]
zfsadm lsaggr: list aggregates
Usage: zfsadm lsaggr [-level] [-help]
```

Related information

Commands:

```
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 name] [-level] [-help]
```

Options

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

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

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

Usage

zfsadm lsaggr displays information about all attached aggregates.

zfsadm lsaggr displays a separate line for each aggregate. Each line displays the following information:
- The aggregate name.
- The name of the system that is the zFS owner of the aggregate. If the aggregate is unowned, `*UNOWNED` is displayed.
- The mode of the aggregate.
- The status of the aggregate (for example, QUIESCED, DISABLED, or both).

You can use the **zfsadm aggrinfo** command to display information about the amount of disk space available on a specific aggregate or on all aggregates on a system.

Privilege required

| If you are using an IOEFSPRM file in your startup proc, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

The following example shows that five aggregates are attached to the system or the sysplex when running in a shared file system environment.

```
zfsadm lsaggr
OMVS.PRV.AGGR004.LDS0004                    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

Commands:
  `zfsadm aggrinfo`
| **`zfsadm fsinfo`**

Files:
  IOEFSPRM

zfsadm lsfs

Purpose

Lists all the file systems on a given aggregate or all attached aggregates.

Format

```
zfsadm lsfs [-aggregate name| -system sysname] [{-fast | -long}] [-level] [-help]
```

Options

-aggregate *name*
: Specifies an aggregate name that is used to retrieve file system information. The aggregate name is not case-sensitive. It is always translated to uppercase. If this option is not specified, the command displays information for all attached aggregates.

-fast
: Causes the output of the command to be shortened to display only the aggregate name.

-help
: Prints the online help for this command. All other valid options that are specified with this option are ignored.

-level
: Prints the level of the **zfsadm** command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

-long
: Causes the output of the command to be extended to display the following additional information about space usage in a file system: the allocation limit, the free space limit, the size of the inode table, the number of file requests, the version of the file system, the creation date and time, and the last update date and time.

-system *sysname*
: Specifies the name of the system that owns the aggregates that contain the file systems to be displayed.

Usage

The **zfsadm lsfs** command displays information about file systems in aggregates. The file systems do not need to be mounted. The **zfsadm lsfs** command displays the following information for a specified aggregate or all attached aggregates on a system or all attached aggregates in the sysplex:

- The total number of file systems that are contained in the aggregate.
- The file system's name (with a `.bak` extension, if appropriate).
- The type (RW for read/write, or BK for backup).
- Whether it is mounted.
- The allocation usage and the free space usage, in kilobytes.
- Whether the file system is online.
- Whether the backup is being deleted.
- The total number of file systems online, offline, busy, and mounted appear at the end of the output for all file systems.

If `-fast` is specified, it only displays the file system names.

If `-long` is specified, the following information is displayed:

- Total number of file systems that are contained in the aggregate.
- File system's name.
- File system's ID.
- The type (RW for read/write, or BK for backup).
- Whether it is mounted or not.
- State vector of the file system.
- Whether the file system is online or not.
- Whether the backup is being deleted.
- Allocation limit and allocation usage.
- Free space limit and free space usage.
- Size of the Filesystem Inode Table and the number of file requests.
- Version of the aggregate.
- Day, date, and time when the file system was created.
- Day, date, and time when the contents of the file system were last updated.
- Total number of file systems online, offline, busy, and mounted appears at the end of the output for all file systems.

Privilege required

If you are using an IOEFSPRM file in your startup proc, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

The following example displays information for the aggregate OMVS.PRV.AGGR001.LDS0001:

```
zfsadm lsfs -aggregate omvs.prv.aggr001.lds0001 -long
IOEZ00129I Total of 1 file systems found for aggregate OMVS.PRV.AGGR001.LDS0001
OMVS.PRV.FS1 100000,,5 RW (Not Mounted)      states 0x10010005 On-line
    4294967232 K alloc limit;          9 K alloc usage
         25000 K quota limit;          9 K quota usage
             8 K Filesystem Inode Table    0 file requests

    version 1.4
    Creation Thu Aug  9 17:17:03 2001
    Last Update Thu Aug  9 17:17:03 2001

Total file systems online 1; total off-line 0; total busy 0; total mounted 0
```

Related information

Commands:

   zfsadm fsinfo

zfsadm lssys

Purpose

Shows the names of the members in a sysplex.

Format

```
zfsadm lssys [-level] [-help]
```

Options

-help Prints the online help for this command. All other valid options specified with this option are ignored.

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

Usage

The **zfsadm lssys** command displays the names of the members in a sysplex.

Privilege required

| If you are using an IOEFSPRM file in your startup proc, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

Examples

The command that follows shows the current list of system names in the XCF group for zFS.

```
zfsadm lssys

IOEZ00361I A total of 3 systems are in the XCF group for zFS
DCEIMGVM
DCEIMGVQ
DCEIMGVN
```

Related information

Related commands:
zfsadm lsaggr

zfsadm query

Purpose

Displays internal zFS statistics (counters and timers) maintained in the zFS Physical File System (PFS).

Format

```
zfsadm query [-system sysname] [-locking] [-reset] [-storage] [-usercache] [-trancache]
             [-iocounts] [-iobyaggregate] [-iobydasd] [-knpfs] [-logcache]
             [-metacache] [-dircache] [-vnodecache] [-ctkc] [-svi] [-stsm]
             [-level] [-help]
```

Options

-ctkc Displays the sysplex client operations report. For more information about this report, see “Statistics Sysplex Client Operations Information” on page 423

-dircache
Displays the directory cache counters report. Beginning in z/OS V1R13, this option is not meaningful; the report will show zeros. For more information about this report, see “Statistics Directory Cache Information” on page 343.

-help Prints the online help for this command. All other valid options that are specified with this option are ignored.

-iobyaggregate
Displays the I/O count by aggregate report. For more information about this report, see “Statistics iobyaggr Information” on page 347.

-iobydasd
Displays the I/O count by Direct Access Storage Device (DASD) report. For more information about this report, see “Statistics iobydasd Information” on page 355.

-iocounts
Displays the I/O count report. For more information about this report, see “Statistics iocounts Information” on page 363.

-knpfs Displays the kernel counters report. 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. For more information about this report, see “Statistics Kernel Information” on page 370.

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

-locking
Displays the locking statistics report. For more information about this report, see “Statistics Locking Information” on page 378.

-logcache
Displays the log cache counters report. For more information about this report, see “Statistics Log Cache Information” on page 387.

-metacache
Displays the metadata cache counters report. For more information about this report, see "Statistics Metadata Cache Information" on page 398.

-reset Resets the report counters to zero. Should be specified with a report type. The reset takes place after displaying the current values. For example, if you enter **`zfsadm query -knpfs -reset`**, the command returns the current values for the kernel counters report before resetting to zero.

| **-stkm** Displays the server token manager report. For more information about this report, see "Statistics Server Token Management Information" on page 405.

-storage
| Displays the storage report. For more information about this report, see "Statistics Storage Information" on page 411.

| **-svi** Displays the server vnode interface statistics report. For more information about this report, see "Statistics Sysplex Owner Operations Information" on page 430.

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

-trancache
| Displays the transaction cache counters report. Beginning with z/OS V2R2, this option is not meaningful; the report will show zeroes. For more information about this report, see "Statistics Transaction Cache Information" on page 438.

-usercache
| Displays the user cache report. For more information about this report, see "Statistics User Cache Information" on page 442.

-vnodecache
Displays the vnode cache counters report. For more information about this report, see "Statistics Vnode Cache Information" on page 454.

Usage

Use the **`zfsadm query`** command to display performance statistics that are maintained by the zFS Physical File System.

Privilege required

| If you are using an IOEFSPRM file in your startup 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.

```
zfsadm query -iobyaggr
zFS I/O by Currently Attached Aggregate
DASD   PAV
VOLSER IOs Mode  Reads       K bytes     Writes      K bytes     Dataset Name
------ --- ----  ----------  ----------  ----------  ----------  ------------
CFC000   1  R/W          13          92        7641       30564  PLEX.JMS.AGGR001.LDS0001
CFC000   1  R/O           9          60           0           0  PLEX.JMS.AGGR002.LDS0002
CFC000   1  R/W          26         188        4483       17952  PLEX.JMS.AGGR004.LDS0004
------ --- ----  ----------  ----------  ----------  ----------  ------------
     3                   48         340       12124       48516  *TOTALS*

Total number of waits for I/O:         52
Average I/O wait time:                 3.886 (msecs)
```

Related information

Commands:

| • **zfsadm fsinfo**
- **zfsadm lsaggr**

zfsadm quiesce

Purpose

Specifies that an aggregate and the file system that is contained in it should be quiesced.

Format

```
zfsadm quiesce {-all | -aggregate name} [-level] [-help]
```

Options

-aggregate *name*
: Specifies the name of the aggregate that is to be quiesced. The aggregate name is not case-sensitive. It is always translated to uppercase. An aggregate must be attached to be quiesced. All current activity against the aggregate is allowed to complete but no new activity is started. Any mounted file systems are quiesced.

-all
: Specifies that all attached aggregates are to be quiesced. Use this option or use `-aggregate`.

-help
: Prints the online help for this command. All other valid options that are specified with this option are ignored.

-level
: Prints the level of the **`zfsadm`** command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

Usage

The **`zfsadm quiesce`** command is used to temporarily drain activity to the aggregate. During this time:

- The aggregate cannot be detached, or grown.
- No activity can occur against mounted file systems.
- If you attempt to unmount a quiesced compatibility mode aggregate, the attempt fails unless you specify unmount force.

The aggregate can be the target of **`lsaggr`**, **`aggrinfo`**, **`lsfs`** (file systems are indicated as busy). While at least one RWSHARE aggregate remains quiesced, message IOEZ00581E is displayed on the zFS owning system's console. Also, if there is at least one task that is waiting for access to the quiesced file system, message IOEZ00830E is displayed.

While an RWSHARE file system is quiesced, the command D OMVS,F displays `QUIESCED` in the PFS EXCP field.

The aggregate is typically quiesced prior to backing up the aggregate. After the backup is complete, the aggregate can be unquiesced.

Privilege required

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

Examples

The following command quiesces the aggregate OMVS.PRV.AGGR001.LDS0001.

```
zfsadm quiesce -aggregate omvs.prv.aggr001.lds0001
```

```
IOEZ00163I Aggregate OMVS.PRV.AGGR001.LDS0001 successfully quiesced
```

Related information

Commands:
zfsadm aggrinfo
| **zfsadm fsinfo**
zfsadm unquiesce

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). The aggregate name is not case-sensitive. It is always translated to uppercase.

-force
: Specifies to change the auditfid to a new zFS auditfid. If the aggregate already contains the new form of the zFS auditfid that you want to change to a different new zFS auditfid (for example, if you copy an aggregate and then rename it, but keep the old aggregate), you must specify `-force` to avoid inadvertently changing the zFS auditfid.

-help
: Prints the online help for this command. All other valid options that are specified with this option are ignored.

-level
: Prints the level of the **zfsadm** command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

-old
: Specifies that the zFS auditfid is set to binary zeros.

Usage

The **zfsadm setauditfid** command sets or resets the zFS auditfid in the aggregate on disk (based on the VOLSER and the cylinder, cylinder, head, head [CCHH] of the first extent of the aggregate). The aggregate must be attached (mounted). If neither `-force` nor `-old` are specified, a standard form auditfid (binary zeros) is changed to the unique form auditfid. If the aggregate already contains the unique form of the zFS auditfid and you want to change it to a different unique zFS auditfid (for example, if you copy an aggregate and then rename it - keeping the old one), you must specify `-force` to avoid inadvertently changing the zFS auditfid. The zFS auditfid is based on the VOLSER and the CCHH of the first extent, unless you specify `-old`. In that case, the zFS auditfid is set to binary zeros.

In a shared file system environment, whether the **zfsadm setauditfid** command is issued from the system owning the zFS aggregate or from a client system, the new auditfid value will only be visible on the zFS owning system. To make it visible on client systems, issue a remount to the same mode.

Privilege required

| If you are using an IOEFSPRM file in your startup proc, the issuer must have READ authority to the data set that contains the IOEFSPRM file. If you are using parmlib (IOEPRMxx), the issuer does not need special authorization.

The user must be UID 0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Examples

```
zfsadm setauditfid -aggregate OMVS.PRV.AGGR001.LDS0001 -force
```

Related information

Commands:
`zfsadm aggrinfo`
`zfsadm format`

Files:
IOEFSPRM

zfsadm unquiesce

Purpose

Makes an aggregate (and the file system that is 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 uppercase. An aggregate must be attached to be unquiesced. All current activity against the aggregate is allowed to resume. Any mounted file systems are unquiesced.

-all
: Specifies that all attached aggregates are to be unquiesced. Use this option or use `-aggregate`.

-help
: Prints the online help for this command. All other valid options that are specified with this option are ignored.

-level
: Prints the level of the **`zfsadm`** command. This option is useful when you are diagnosing a problem. Except for `-help`, all other valid options that are specified with `-level` are ignored.

Usage

The **`zfsadm unquiesce`** command allows activity that was suspended by **`zfsadm quiesce`**, to be resumed.

The aggregate is typically quiesced prior to backing up the aggregate. After the backup is complete, the aggregate can be unquiesced and the backup change activity flag can be reset.

Privilege required

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

Examples

The following command unquiesces the aggregate OMVS.PRV.AGGR001.LDS0001.

```
zfsadm unquiesce -aggregate omvs.prv.aggr001.lds0001

IOEZ00166I Aggregate OMVS.PRV.AGGR001.LDS0001 successfully unquiesced
```

Related information

Commands:
zfsadm aggrinfo
| **zfsadm fsinfo**
zfsadm quiesce

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

This section describes the IOEFSPRM file, which is a data set that is used during zFS processing.

IOEFSPRM

Purpose

The IOEFSPRM file lists the configuration options for the zFS PFS and the batch utilities **ioefsutl** and **ioeagslv**. There is no mandatory information in this file; therefore, it is not required. The options all have defaults. However, if you need to specify any options (for tuning purposes, for example), you must have an IOEFSPRM file.

zFS allows for more than one method to specify the location of the IOEFSPRM configuration file. These methods are described in this chapter. zFS uses the following criteria to determine which method to use:

- If an IOEZPRM DD statement exists in the JCL, the data set that it defines will be the configuration file for the local system.
- If there is no IOEZRPM DD statement, the IOEPRMxx parmlib members that are specified in the PARM string of the zFS FILESYSTYPE statement is used.
- If there is no PARM string on the zFS FILESYSTYPE statement, parmlib member IOEPRM00 is used.
- If there is no IOEPRM00 parmlib member, no zFS configuration data set will be used.

The location of the IOEFSPRM file can be specified by the IOEZPRM DD statement in the ZFS PROC and in the JCL for the **ioefsutl** or **ioeagslv** batch utilities. (See "Terminology and concepts" on page 4 for a definition of the term "ZFS PROC.") However, the preferred method for specifying the zFS configuration option file is to use the IOEPRMxx parmlib member as described in "Using PARMLIB (IOEPRMxx)" on page 221. If you still want to use a single IOEFSPRM file, specify the IOEZPRM DD statement in your JCL. The IOEFSPRM file is typically a PDS member, so the IOEZPRM DD statement might look like the following example:

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

If you need to have separate IOEFSPRM files and you want to share the ZFS PROC in a sysplex, you can use a system variable in the ZFS PROC so that it points to different IOEFSPRM files. The IOEZPRM DD might look like the following:

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

Your IOEFSPRM file might reside in SYS4.PVT.SY1.PARMLIB(IOEFSPRM) on system SY1; in SYS4.PVT.SY2.PARMLIB(IOEFSPRM) on system SY2; and others.

If you want to share a single IOEFSPRM file, you can use system symbols in data set names in the IOEFSPRM file. For example, `msg_output_dsn=USERA.&SYSNAME..ZFS.MSGOUT` results in `USERA.SY1.ZFS.MSGOUT` on system SY1. Each system has a single (possibly shared) IOEFSPRM file.

Any line beginning with # or * is considered a comment. The text in the IOEFSPRM file is not case-sensitive. Any option or value can be uppercase or lowercase. Blank lines are allowed. Do not have any sequence numbers in the

IOEFSPRM file. If you specify an invalid text value, the default value is assigned. If you specify an invalid numeric value, and it is smaller than the minimum allowed value, the minimum value is assigned. If you specify an invalid numeric value, and it is larger than the maximum allowed value, the maximum value is assigned.

Using PARMLIB (IOEPRMxx)

The preferred alternative to a IOEZPRM DDNAME is specifying the IOEFSPRM file as a parmlib member. In this case, the member has the name IOEPRM*xx*, where *xx* is specified in the parmlib member list.

When the IOEFSPRM is specified in a DD statement, 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 parmlib files. This allows an installation to specify configuration options that are common among all members of the sysplex (for example, `adm_threads`) in a shared IOEPRMxx member and configuration options that are system-specific (for example, `trace_dsn`) in a separate, system-specific IOEPRMxx member. If a configuration option is specified more than once, the first one found is taken. For more information about the IOEPRMxx parmlib member, *z/OS MVS Initialization and Tuning Reference*.

The IOEPRMxx files are contained in the logical parmlib concatenation. The logical parmlib concatenation is a set of up to ten partitioned data sets defined by parmlib statements in the LOADxx member of either SYSn.IPLPARM or SYS1.PARMLIB. The logical parmlib concatenation contains zFS IOEPRMyy members that contain zFS configuration statements. Columns 72-80 are ignored in the IOEPRM*yy* member. The *yy* values are specified in the PARM option of the FILESYSTYPE statement for the zFS PFS (in the BPXPRMxx parmlib member). The only valid value that can be specified on the PARM option for the zFS PFS is the parmlib search parameter `PRM=`. The PARM string is case-sensitive. As the following example shows, you must enter the string in uppercase.

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

The parmlib concatenation can also be specified in the **`ioeagslv`** and **`ioefsutl`** batch utility parameters. Specify the `-PRM` keyword in the PARM string on the EXEC statement to use IOEPRMxx parameter file members. For more information, see “ioeagslv” on page 130 and “ioefsutl” on page 136.

Up to 32 member suffixes can be specified. You can also use any system symbol that resolves to two characters.

```
FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSCM)
ASNAME(ZFS,'SUB=MSTR')
PARM('PRM=(01,&SYSCLONE.)')
```

See Figure 43 on page 135 for an example of using PRM.

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.

To specify 32 members, type the member suffixes up to column 71; then, continue them in column 1 on the next line, as shown in Figure 50.

```
                                                                           col 72
                                                                              |
                                                                              ^
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)')
^
|
col 1
```

Figure 50. How to specify 32 members

If no PRM suffix list is specified (and no IOEZPRM DD is specified in their respective JCL), then parmlib member IOEPRM00 is read. Parmlib support is only used when no IOEZPRM DD is present in the JCL.

IOEFSPRM and IOEPRMxx

Descriptions of the valid configuration variables and their respective allowed values follow. If no IOEFSPRM file is found, the default values for each configuration value are used.

Processing options for IOEFSPRM and IOEPRMxx

The following processing options are used for the zFS PFS.

adm_threads
Specifies the number of threads that are defined to handle pfsctl or mount requests.

Default value: 10

Expected value: A number in the range 1 - 256.

Example: `adm_threads=5`

aggrfull
Specifies the threshold and increment for reporting aggregate utilization messages to the operator. The `aggrfull` parameter is independent of `fsfull`. However, `aggrfull` reports are based on free 8-K blocks; while `fsfull` reports are based on free 1 K blocks. The `aggrfull` value tends to give a more accurate view of free space and is the recommended choice.

Default value: Off

Expected value: Two numbers in the range 1 - 99 within parentheses that are separated by a comma.

Example: `aggrfull(90,5)`

aggrgrow
Specifies whether aggregates can be dynamically extended when they become full. As of z/OS V1R13, by default, a zFS read/write mounted file system that is mounted on a system running z/OS V1R13 or later attempts to dynamically extend when it runs out of space. The aggregate (that is, the VSAM linear data set) must have a secondary allocation that is specified to be dynamically extended and there must be space on the volumes. This global value can be overridden on the MOUNT command for compatibility mode aggregates. For an explanation of the rules for extending a VSAM LDS, see *z/OS DFSMS Using Data Sets*.

Default value: On

Expected value: On or Off

Example: `aggrgrow=on`

change_aggrversion_on_mount
Specifies whether a version 1.4 aggregate should be changed to a version 1.5 aggregate on a primary read/write mount. No directories are converted to extended (v5) directories. The CONVERTTOV5 or NOCONVERTTOV5 MOUNT PARM overrides this option.

Default value: Off

Expected value: On or Off

Example: `change_aggrversion_on_mount=off`

client_reply_storage
Specifies the amount of storage that is used to handle sysplex server replies.

Default value: 10 M

Expected value: A number in the range 2M - 128M. K or M can qualify the number.

Example: `client_reply_storage=8M`

convert_auditfid
Specifies whether the zFS auditfid of an aggregate is automatically converted from the old form auditfid (binary zeros) to the new form auditfid on a read/write mount (attach). If the auditfid is already the new form, it is not changed. An auditfid of the new form will cause zFS to generate new auditids for files and directories in the file system.

Default value: On

Expected value: On or Off

Example: `convert_auditfid=on`

converttov5
Specifies whether a zFS read/write file system is assigned the converttov5 attribute. If it is assigned the converttov5 attribute and the aggregate is a version 1.5 aggregate, zFS will automatically convert directories from v4 to extended (v5) as they are accessed. If the converttov5 attribute is assigned

at primary mount time, a version 1.4 aggregate will be changed to a version 1.5 aggregate. The CONVERTTOV5 or NOCONVERTTOV5 MOUNT PARM overrides this option.

If automatic directory conversion for a directory fails, it is not attempted again until the file system is unmounted and mounted again.

Default value: Off

Expected value: On or Off

Example: `converttov5=off`

file_threads
Specifies the number of threads that handle sysplex server requests.

Default value: 32

Expected value: A number in the range 1 - 256.

Example: `file_threads=50`

format_aggrversion
Specifies the default version of an aggregate when formatting it. Each method for formatting a zFS aggregate gets this value from the zFS PFS if no version is specified.

Default value: 4

Expected value: 4 (meaning format a version 1.4 aggregate) or 5 (meaning format a version 1.5 aggregate)

Example: `format_aggrversion=4`

fsfull Specifies the threshold and increment for reporting file system utilization messages to the operator. The `fsfull` parameter is independent of `aggrfull`. Whereas `aggrfull` reports are based on free 8-KB blocks, `fsfull` reports are based on free 1-KB blocks. The `aggrfull` parameter tends to give a more accurate view of free space and is the recommended choice.

Default value: Off

Expected value: Two numbers in the range 1 - 99 within parentheses and separated by a comma.

Example: `fsfull(85,5)`

group Specifies the XCF group that zFS uses to communicate between sysplex members. The `Expected value` characters must be acceptable to XCF. Generally, the characters A-Z, 0-9 and the national characters ($, # and @) are acceptable. The value that is specified must match on all systems in the sysplex that participate in a shared file system environment. Normally, there is no reason to specify this option. For more details, see the GRPNAME parameter of the IXCJOIN macro in *z/OS MVS Programming: Sysplex Services Reference*.

Default value: IOEZFS

Expected value: 1 to 8 characters

Example: `group=IOEZFS1`

log_cache_size

Specifies the size of the cache that is used to contain buffers for log file pages. You can also specify a fixed option, which indicates that the pages are permanently fixed for performance. The `fixed` option reserves real storage for usage by zFS only.

Default value: 16 M

Expected value: 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 that is used to contain metadata. You can also specify a fixed option, which indicates that the pages are permanently fixed for performance. The `fixed` option reserves real storage for usage by zFS only

If `metaback_cache_size` is specified, the size of the entire metadata cache will be a combination of the two values. It is not required, but it is recommended to keep your IOEFSPRM configuration file clean of outdated specifications for simplicity. Therefore, IBM recommends not to use the `metaback_cache_size` option. Rather, the size of the entire metadata cache should be assigned to the `meta_cache_size` option.

zFS provides a check to see if the metadata cache size is less than the calculated default metadata cache size. For more information, see ZFS_VERIFY_CACHESIZE in *IBM Health Checker for z/OS User's Guide*.

Default value: If `metaback_cache_size` is specified, then `meta_cache_size` is 64 M. If `metaback_cache_size` is not specified, zFS calculates 10% of real storage that the system has available during zFS initialization.

- If this amount is less than 64 M, then `meta_cache_size` is assigned 64 M.
- If this amount is between 64 M and 2 G+100 M, then `meta_cache_size` is assigned 10% of real storage size.
- If the amount is greater than 2 G+100 M, then `meta_cache_size` is assigned 2 G+100 M

Expected value: A number in the range 1 M - 64 G. A K or M or G can be appended to the value to mean kilobytes, megabytes, or gigabytes, respectively.

Example: `meta_cache_size=64M,fixed`

metaback_cache_size

Specifies the size of the backing portion of the metadata cache. The backing cache is no longer in a data space. Rather, it is combined with `meta_cache_size` into one cache with a size of the sum of the two values.

Tip: To avoid confusion, do not keep outdated specifications in your IOEFSPRM configuration file. Use only the `meta_cache_size` option to specify the entire size of the metadata cache.

zFS provides a check to see if the sum of the metadata cache size and metadata backing cache size is less than the sum of the default metadata cache size and metadata backing cache size. For more information, see ZFS_VERIFY_CACHESIZE in *IBM Health Checker for z/OS User's Guide*.

zFS provides a check to indicate whether this configuration option is specified. For more information, see ZFS_CACHE_REMOVALS in *IBM Health Checker for z/OS User's Guide*

Default value: If `meta_cache_size` is specified, then there is no value for the metaback cache. Otherwise, see the default calculation description in `meta_cache_size`.

Expected value: A number in the range 1 M - 2048 M. A K or M can be appended to the value to mean kilobytes or megabytes, respectively.

Example: `metaback_cache_size=64M`

modify_cmd_threads
Specifies the number of threads that are defined to handle zFS modify commands.

Default value: 3

Expected value: A number in the range 1 - 256.

Example: `modify_cmd_threads=1`

quiesce_message_delay
Specifies the minimum number of seconds to delay issuing the IOEZ00830E message after it is determined that there is at least one quiesced aggregate and it needs to be displayed.

Default value: 30

Expected value: A number in the range 30 - 21474836.

Example: `quiesce_message_delay=300`

quiesceinfo_message_delay
Specifies the minimum number of seconds to delay issuing the IOEZ00581E message after it is determined that there is at least one task waiting to access a quiesced aggregate and it needs to be displayed.

Default value: 30

Expected value: A number in the range 30 - 21474836.

Example: `quiesceinfo_message_delay=300`

romount_recovery
Specifies whether zFS will automatically avoid a read-only mount failure because of the need to run log recovery for this aggregate. This can occur when the aggregate has been mounted read/write, and then a failure occurs before it was unmounted. If the next mount is for read-only, log recovery must run for the mount to be successful. When this situation

occurs and `romount_recovery=on`, zFS temporarily mounts the aggregate read/write to run log recovery, and then zFS unmounts and mounts the aggregate read-only.

Default value: Off

Expected value: On or Off

Example: `romount_recovery=on`

recovery_max_storage

Indicates the maximum amount of zFS address space storage to use for concurrent log recovery during multiple concurrent aggregate mounts (attaches). This allows multiple concurrent mounts to occur when sufficient storage is available for multiple concurrent log recovery processing.

Default value: 256 M

Expected value: A number in the range 128 M - 512 M.

Example: `recovery_max_storage=128M`

sync_interval

Specifies the number of seconds between syncs.

Default value: 30

Expected value: A number in the range 11 - 21474836.

Example: `sync_interval=45`

sysplex

Starting with z/OS V1R13, zFS always runs sysplex-aware by file system, regardless of the sysplex specification. If you specify `sysplex=on`, zFS changes the default of `sysplex_filesys_sharemode` to `rwshare`. Otherwise, the default for `sysplex_filesys_sharemode` is `norwshare`. If you specify `sysplex=off`, the result is the same as specifying `sysplex=filesys`. For information about whether to make a read/write file system sysplex-aware, see "Using zFS read/write sysplex-aware file systems" on page 17.

Default value: `filesys`

Expected value: `Off`, `filesys`, or `On`, if BPXPRMxx specifies SYSPLEX(YES).

| Ignored, if BPXPRMxx does not specify SYSPLEX(YES).

Tip: Specify `sysplex=filesys`.

Example: `sysplex=filesys`

sysplex_filesys_sharemode

Specifies the default for the mount PARM for a zFS read/write file system
| that is mounted in a shared file system environment. For information about whether to make a read/write file system sysplex-aware, see "Using zFS read/write sysplex-aware file systems" on page 17.

Default value: `norwshare` (unless `sysplex=on` was specified, then the default is `rwshare`)

Expected value: `rwshare` or `norwshare`

Example: `sysplex_filesys_sharemode=rwshare`

token_cache_size
Specifies the maximum number of tokens in the server token manager cache to use for cache consistency between zFS members. The number of tokens that are initially allocated for the server token manager cache is 20480.

Default value: Double the number of vnodes (see vnode_cache_size) when running in a shared file system environment. If you are not running in a shared file system environment, then there is no default value. This option is meaningful only when zFS is running sysplex-aware.

Expected value: A number in the range 20480 - 20 million.

Example: `token_cache_size=30720`

user_cache_size
Specifies the size, in bytes, of the cache that is used to contain file data. You can also specify a fixed option, which indicates that the pages are permanently fixed for performance. The `fixed` option reserves real storage for usage by zFS only.

zFS provides a check to see if the user cache size is less than the default user cache size. For more information, see ZFS_VERIFY_CACHESIZE in *IBM Health Checker for z/OS User's Guide*.

Default value: zFS calculates 10% of real storage the system has available during zFS initialization. If this amount is less than 256 M, then the default is 256 M. If this amount is between 256 M and 2 G, then the default is 10% of real storage. If the amount is greater than 2 G, then the default is 2 G.

Expected value: A number in the range 10 MB - 65536 MB (64 G). K or M can be appended to the value to mean kilobytes or megabytes.

Example: `user_cache_size=64M,fixed`

user_running_hangdump
Specifies whether a hang dump should be taken for a user task that has been hanging for approximately 5 minutes.

Default value: Off

Example: `user_running_hangdump=on`

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 1000 to 10 million.

Example: `vnode_cache_size=131072`

The following options are used during debugging of the zFS PFS, the batch utilities (**`ioeagfmt`**, **`ioeagslv`**, and **`ioefsutl`**) and the **`zfsadm`** command. They might not apply to the utilities and commands that are listed in the preceding section.

cmd_trace
Specifies whether command tracing is done for the **`ioeagfmt`** batch utility or a **`zfsadm`** command. If On, a zFS trace will be printed in the data set specified by the zFS PFS `trace_dsn` configuration option after the batch utility or command completes. A trace from **`ioeagfmt`** has a member name of IOEAGT01. A trace from a **`zfsadm`** command has a member name of ZFSADT01.

Default value: Off

Expected value: On or Off.

Example: `cmd_trace=on`

debug_settings_dsn
Specifies the name of a data set containing debug classes to enable when the zFS PFS or the batch utilities start. It is read when zFS is started (or restarted). The debug classes are also used by the batch utilities.

Default value: None

Expected value: The name of a data set containing debug classes to enable.

Example: `debug_settings_dsn=usera.zfs.debug.input(file1)`

max_errors
The maximum number of errors that the salvager program allows before it stops. If this limit is exceeded, the salvager program ends with message IOEZ00752E.

Default value: 100000

Expected value: A number in the range 1000 - 1000000

Example: `MAX_ERRORS=5000`

msg_input_dsn
Specifies the name of a data set containing translated zFS messages. It is specified when the installation uses messages that are in languages other than English. (When you use English messages, do not specify this option.) It is read when zFS or the batch job is started (or restarted). Currently, Japanese messages are supported.

Default value: None

Expected value: The name of a data set containing translated zFS messages.

Example: `msg_input_dsn=usera.sioemjpn`

msg_output_dsn
Specifies the name of a data set that contains any output messages that

come from the zFS PFS during initialization. See Chapter 8, “Performance and debugging,” on page 67. This is not a required parameter.

Default value: None

Expected value: The name of a data set that contains zFS PFS messages issued.

Example: `msg_output_dsn=usera.zfs.msg.out`

trace_dsn

Specifies the name of a data set that contains the output of any operator MODIFY ZFS,TRACE,PRINT commands or the trace output if the zFS PFS or the batch utilities abends. Each trace output creates a member in the PDSE. Traces that come from the zFS PFS kernel have member names of ZFSKNT*nn*. Traces from the salvager program have member names of ZFSSLV*nn*. Traces that come from the **`ioefsutl`** program have member names that start with FSUTL*nn*.*nn* starts with 01 and increments for each trace output. *nn* is reset to 01 when zFS is started (or restarted). See Chapter 8, “Performance and debugging,” on page 67. This is not a required parameter. If it is not specified, only a dump is generated if an abend occurs.

Default value: None

Expected value: The name of a PDSE data set.

Example: `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 and the salvager address space that is used for internal tracing that is always on. The trace can be sent to the `trace_dsn` by using the operator MODIFY ZFS,TRACE,PRINT
| command. You can set the `trace_table_size` up to 65535 M, but to print the trace to a PDSE you must limit its size to 750 M.

Default value:
- 16 M for the zFS address space
- 64 M for the salvager address space

| **Expected value:** A number in the range 1 M - 65535 M.

| **Example:** `trace_table_size=256M`

user_running_hangdump

Specifies that if a user task appears to be hung for approximately 5 minutes, a dump of the user address space is obtained by the ZFS hang detector. This dump is with abend code 2C3 and reason code EA5805DB. This dump is accompanied by message IOEZ00605I. Use this message description to diagnose the problem.

Default value: Off

Expected value: On or Off

Example: `user_running_hangdump=ON`

xcf_trace_table_size

Specifies the size of the XCF trace table.

Default value: 4 M

Expected value: A number in the range 1 M - 65535 M

Example: `xcf_trace_table_size=8M`

Examples

Following is a sample IOEFSPRM file that contains program options.

```
**********************************************************************
* zFS Sample Parameter File:  IOEFSPRM
* For a description of these and other zFS parameters, refer to the
* zFS Administration document.
* Notes:
*  1. The IOEFSPRM file and parameters in the file are optional but it
*     is recommended that the parameter file be created in order to be
*     referenced by the DDNAME=IOEZPRM statement the PROCLIB JCL for
*     the zFS started task or through the IOEPRMxx parmlib member.
*  2. An asterisk in column 1 identifies a comment line.
*  3. A parameter specification must begin in column 1.
************************************************************************
* The following msg_output_dsn parameter defines the optional output
* message data set. If this parameter is not specified, or if the data
* set is not found, messages will be written to the system log.
* You must delete the * from a line to activate the parameter.
************************************************************************
*msg_output_dsn=usera.zfs.msg.out
************************************************************************
* The following msg_input_dsn parameter is ONLY required if the optional
* NLS feature is installed. The parameter specifies the
* message input data set containing the NLS message text which is
* supplied by the NLS feature. If this parameter is not specified or if
* the data set is not found, English language messages will be generated
* by zFS. You must delete the * from a line to activate the parameter.
************************************************************************
*msg_input_dsn=usera.sioemjpn
************************************************************************
* The following are examples of some of the optional parameters that
* control the sizes of caches, tuning options, and program operation.
* You must delete the * from a line to activate a parameter.
************************************************************************
*adm_threads=5
*aggrfull(90,5)
*aggrgrow=on
*change_aggrversion_on_mount=off
*client_reply_storage=10M
*cmd_trace=off
*convert_auditfid=off
*converttov5=off
*file_threads=40
*format_aggrversion=4
*fsfull(85,5)
*group=IOEZFS1
*log_cache_size=32M
*meta_cache_size=64M
*romount_recovery=off
*recovery_max_storage=128M
*sync_interval=45
*sysplex=filesys
*sysplex_filesys_sharemode=norwshare
*token_cache_size=65536
```

```
*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=256M
*xcf_trace_table_size=8M
```

Chapter 13. zFS application programming interface information

Information is presented about the zFS commands and their respective subcommands. They can be used to manage zFS aggregates and file systems, and to query or set configuration options. Following is a list of the zFS commands:

- ZFSCALL_AGGR (0x40000005)
- ZFSCALL_CONFIG (0x40000006)
- ZFSCALL_FILESYS (0x40000004)
- ZFSCALL_FSINFO (0x40000013)
- ZFSCALL_STATS (0x40000007)

The z/OS UNIX **pfsctl** (command X'C000000B') can also retrieve zFS reason code text. For more information, see the description of the PC#ErrorText **pfsctl** command in the usage notes for the BPX1PCT service in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

For information about how to invoke the pfsctl (BPX1PCT) application programming interface in a 64-bit environment, refer to Appendix A, “Running the zFS pfsctl APIs in 64-bit mode,” on page 467.

This topic also describes a zFS w_pioctl call for **fileinfo**.

pfsctl (BPX1PCT)

Purpose

The pfsctl (BPX1PCT) application programming interface is used to send requests to a physical file system. It is documented in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*. zFS is a physical file system and supports several zFS-specific pfsctl functions, which are documented in this section.

Format

```
BPX1PCT (File_system_type,
         Command,
         Argument_Length,
         Argument,
         Return_value,
         Return_code,
         Reason_code);
```

Parameters

File_system_type
An eight-character field. In the case of zFS, it contains the characters ZFS, followed by five blanks.

Command
| An integer. There are five major ZFS commands:
- ZFSCALL_AGGR (0x40000005)
- ZFSCALL_CONFIG (0x40000006)
- ZFSCALL_FILESYS (0x40000004)
- | ZFSCALL_FSINFO (0x40000013)
- ZFSCALL_STATS (0x40000007)

Each command has a set of subcommands.

Argument_Length
An integer that contains the length of the argument.

Argument
A structure that has the pfsctl parameters followed by the subcommand parameters. The definitions of any structures that have padding bytes added by the compiler, have the padding bytes explicitly declared in the examples.

| The fields of the structures are described in the Format sections of each
| API. These descriptions contain structure names, field names inside the
| structures, the length of the field, and a brief description of what the field
| is used for. The lengths of the field names contain C types and are as
| follows:

- | `int` or `unsigned int` are four bytes.
- | `long long`, `unsigned long long`, `long long int`, and `unsigned long long`
  | `int` are 8 bytes.

The following list shows the general format of the **`Argument`** for all subcommands, where *n* depends on the particular subcommand:

```
Subcommand operation code     int
Parameter0                    int
Parameter1                    int
Parameter2                    int
Parameter3                    int
Parameter4                    int
Parameter5                    int
Parameter6                    int
Buffer[n]                     char[n]
```

Return_value
An integer that contains `0` if the request is successful or `-1` if it is not successful.

Return_Code
An integer in which the return code is stored. See *z/OS UNIX System Services Messages and Codes* for these codes.

Reason_Code
An integer that stores the reason code. If this code is of the form `0xEFnnxxxx`, see *z/OS Distributed File Service Messages and Codes*. Otherwise, see *z/OS UNIX System Services Messages and Codes*.

Usage notes

| The major commands are summarized in Table 17 and described in detail in the following sections. The zFS pfsctl APIs will work across sysplex members. That is, zFS pfsctl APIs can query and set information on zFS aggregates that are owned by the current system. They can also access and set file system information from other systems in the sysplex.

The z/OS UNIX pfsctl (command X'C000000B') can also retrieve zFS reason code text. For more information, see the description of the PC#ErrorText pfsctl command in the usage notes for the BPX1PCT service in *z/OS UNIX System Services Programming: Assembler Callable Services Reference*.

Table 17. Summary of APIs for zFS pfsctl

Command	Code	Subcommands and opcodes
Aggregate	ZFSCALL_AGGR (0x40000005)	• Attach Aggregate (105) • 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 Auditfid (149) • Unquiesce Aggregate (133) • Reset backup flag (157)
File System	ZFSCALL_FILESYS (0x40000004)	• List File System Status (142)

Table 17. Summary of APIs for zFS pfsctl (continued)

Command	Code	Subcommands and opcodes
Configuration	ZFSCALL_CONFIG (0x40000006)	• List Systems (174) • Query adm_threads setting (180) • Query aggrfull setting (181) • Query aggrgrow setting (182) • Query change_aggrversion_on_mount (246) • Query client_cache_size (231) • Query client_reply_storage (223) • Query cmd_trace (184) • Query convert_auditfid (237) • Query converttov5 (250) • Query debug_settings_dsn setting (186) • Query file_threads (217) • Query format_aggrversion (248) • Query fsfull setting (187) • Query group setting (214) • Query log_cache_size setting (193) • Query meta_cache_size setting (198) • Query metaback_cache_size setting (199) • Query modify_cmd_threads (251) • Query msg_input_dsn setting (200) • Query msg_output_dsn setting (201) • Query sync_interval setting (205) • Query syslevel (238) • Query sysplex_filesys_sharemode (244) • Query sysplex_state (215) • 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_size setting (210) • Query vnode_cache_size setting (212) • Set adm_threads (150) • Set aggrfull (158) • Set aggrgrow (171) • Set change_aggrversion_on_mount (245) • Set client_cache_size (230) • Set client_reply_storage (222) • Set convert_auditfid (236) • Set converttov5 (249) • Set file_threads (176) • Set format_aggrversion (247) • Set fsfull (157) • Set log_cache_size (153) • Set meta_cache_size (152) • Set metaback_cache_size (163) • Set modify_cmd_threads (173) • Set msg_output_dsn (161) • Set sync_interval (154) • Set sysplex_filesys_sharemode (243) • Set token_cache_size (177) • Set trace_dsn (159) • Set tran_cache_size (160) • Set user_cache_size (151) • Set vnode_cache_size (155)

Table 17. Summary of APIs for zFS pfsctl (continued)

Command	Code	Subcommands and opcodes
Statistics	ZFSCALL_STATS (0x40000007)	• 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) • Statistics server token management information (252) • Statistics client vnode operations (253) • Statistics server vnode operations (254)
File System Information	ZFSCALL_FSINFO (0x40000013)	• List detailed file system information (153) • Reset file system statistics (154)

Table 18. Summary of zFS w_pioctl calls

Command	Code
fileinfo	0x0000A901

Attach Aggregate

Purpose

This subcommand call is an aggregate operation that attaches an aggregate to a system. This makes the aggregate and all its file systems known to the zFS physical file system running on that system. (Compatibility mode aggregates are attached during mount so that a separate attach is not necessary.)

Format

```
syscall_parmlist
   opcode           int              105       AGOP_ATTACH_PARMDATA
   parms[0]         int              offset to AGGR_ID
   parms[1]         int              offset to AGGR_ATTACH
   parms[2]         int              offset to system name (optional)
   parms[3]         int              0
   parms[4]         int              0
   parms[5]         int              0
   parms[6]         int              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          char             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    No longer supported
     ATT_GROW                        0x04    Allow dynamic grow
     ATT_NOGROW                      0x02    Disallow dynamic grow
   at_res2          char                0
   at_reserved      int[64]             0 reserved for future use
systemname          char[9]

Return_value    0 if request is successful, -1 if it is not successful

Return_code
  EEXIST        Aggregate already attached
  EINTR         ZFS is shutting down
  EMVSERR       Internal error using an osi service
  EPERM         Permission denied to perform request
  EINVAL        Attempt to attach a multi-file system aggregate

Reason_code
  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage notes

- The ATT_NBS and ATT_NONBS flags are no longer supported; zFS always runs with NBS on. If either of these parameters is specified, it is ignored.

- ATT_GROW and ATT_NOGROW are mutually exclusive. If neither is specified, the default is the aggrgrow setting in the IOEFSPRM file. See "Dynamically growing a compatibility mode aggregate" on page 30 for a description of dynamic grow.
- The at_threshold and at_increment values are ignored unless ATT_MONITOR is set.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

Detach Aggregate

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

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   /* no longer supported */
#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 */
} AGGR_ATTACH;

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;
  struct parmstruct myparmstruct;
  char   aggrname[45] = "PLEX.DCEIMGQX.FS"; /* aggregate name to attach */

  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 one  */
  /* 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;

  /* Ensure reserved fields are 0 */
  memset(idp, 0, sizeof(AGGR_ID));
  memset(atp, 0, sizeof(AGGR_ATTACH));
  memset(asp, 0, sizeof(myparmstruct.systemname));

  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);
  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"); */

 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 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;
}
```

Define Aggregate

Purpose

An aggregate operation that defines (creates) a VSAM linear data set, which can then be formatted as a zFS aggregate.

Format

```
syscall_parmlist
   opcode                int            139       AGOP_DEFINE_PARMDATA
   parms[0]              int            Offset to AGGR_DEFINE
   parms[1]              int            Size of Buffer
   parms[2]              int            Offset to Buffer
   parms[3]              int            Offset to system name (optional)
   parms[4]              int            0
   parms[5]              int            0
   parms[6]              int            0
AGGR_DEFINE
   eye                   char[4]        "AGDF"
   len                   short          sizeof(AGGR_DEFINE)
   ver                   char           1
   aggrName              char[45]       Name of aggregate dataset to create
   dataClass             char[9]        Name of a data class
   managementClass       char[9]        Name of a management class
   storageClass          char[9]        Name of a storage class
   model                 char[45]       Name of a model
   modelCatalog          char[45]       Name of a model catalog
   catalog               char[45]       Name of a catalog
   volumes[59]           char[7]        Null terminated list of VOLSERs
   reservedChars1        char           Reserved
   numVolumes            int            Number of volumes to use
   spaceUnit             int            Units space is allocated in
   spacePrimary          unsigned int   Primary allocation
   spaceSecondary        unsigned int   Secondary allocation
   reservedIntsl         int[32]        Reserved space for future use
systemname               char[9]        System name where DEFINE should run

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
  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.
- | Output buffer is space for IDCAMS to return error messages.

Privilege required

The issuer must have sufficient authority to create the VSAM linear data set.

Related services

Format Aggregate

Restrictions

The VSAM linear data set to be defined cannot already exist.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_DEFINE_PARMDATA 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_AGGRNAME 44
#define ZFS_MAX_SMSID 8
#define ZFS_MAX_VOLID 6

typedef struct aggr_define_t {
    char        eye[4];               /* Eye catcher */
#define         ADEF_EYE "AGDF"
    short       len;                  /* Length of this structure */
    char        ver;                  /* Version */
#define         ADEF_VER_INITIAL 1    /* Initial version */
    char        aggrName[ZFS_MAX_AGGRNAME+1];
    char        dataClass[ZFS_MAX_SMSID+1];
    char        managementClass[ZFS_MAX_SMSID+1];
    char        storageClass[ZFS_MAX_SMSID+1];
    char        model[ZFS_MAX_AGGRNAME+1];
    char        modelCatalog[ZFS_MAX_AGGRNAME+1];
    char        catalog[ZFS_MAX_AGGRNAME+1];
    char        volumes[59][ZFS_MAX_VOLID+1];
    char        reservedChars1;
    int         numVolumes;
    int         spaceUnit;
#define         ZFS_SPACE_CYLS 1
#define         ZFS_SPACE_KILO 2
#define         ZFS_SPACE_MEGA 3
#define         ZFS_SPACE_RECS 4
#define         ZFS_SPACE_TRKS 5
    unsigned int spacePrimary;
    unsigned int spaceSecondary;
    int         reservedInts1[32];
} AGGR_DEFINE;

struct parmstruct {
    syscall_parmlist myparms;
    AGGR_DEFINE      aggdef;
    char             Buffer[1024];
    char             systemname[9];
};

int main(int argc, char **argv)
{
    int  bpxrv;
    int  bpxrc;
    int  bpxrs;
    char aggrname[45] = "PLEX.DCEIMGQX.LDS"; /* aggregate name to define */
    char dataclass[9]        = "";
    char managementclass[9]  = "";
    char storageclass[9]     = "";
    char model[45]           = "";
    char modelcatalog[45]    = "";
    char catalog[45]         = "";
    char volumes[7]          = "CFC000";

    struct parmstruct myparmstruct;
    AGGR_DEFINE       *agp           = &(myparmstruct.aggdef);
    char              *bufp          = &(myparmstruct.Buffer[0]);

    /* This next field should only be set if parms[3] is non-zero */
    /* strcpy(myparmstruct.systemname,"DCEIMGVN"); */
    /* set system to run define on */
    myparmstruct.myparms.opcode = AGOP_DEFINE_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, dataclass);
    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,
            &bpxrv,
            &bpxrc,
            &bpxrs);

    if (bpxrv < 0)
    {
        printf("define: Error defining LDS %s\n", aggrname);
        printf("define: BPXRV = %d BPXRC = %d BPXRS = %x\n",
               bpxrv, bpxrc, bpxrs);
        printf("define: job output:\n\n%s\n", myparmstruct.Buffer);
        return bpxrc;
```

```
        }
        else
            printf("define: LDS %s defined successfully\n", aggrname);
        return 0;
    }
```

Detach Aggregate

Purpose

Detach Aggregate is an aggregate operation that detaches an attached, but not mounted, compatibility mode aggregate. Mounted compatability mode aggregates are detached during unmount.

Format

```
syscall_parmlist
   opcode          int           104       AGOP_DETACH_PARMDATA
   parms[0]        int           offset to AGGR_ID
   parms[1]        int           0
   parms[2]        int           0
   parms[3]        int           0
   parms[4]        int           0
   parms[5]        int           0
   parms[6]        int           0
AGGR_ID
   aid_eye         char[4]       "AGID"
   aid_len         char          sizeof(AGGR_ID)
   aid_ver         char          1
   aid_name        char[45]      "OMVS.PRV.AGGR001.LDS0001"
   aid_reserved    char[33]      0

Return_value    0 if request is successful, -1 if it is not successful

Return_code
  EBUSY         Aggregate could not be detached due to mounted file system
  EINTR         ZFS is shutting down
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request

Reason_code
  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

Attach Aggregate

Restrictions

All file systems in the aggregate must be unmounted before the aggregate can be detached.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

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;
    int              bpxrs;
    char             aggrname[45] = "PLEX.DCEIMGQX.FS";
    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;

    /* Ensure reserved fields are 0 */
    memset(&myparmstruct.aggr_id, 0, sizeof(AGGR_ID));

    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 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

Format Aggregate is an aggregate operation that formats a VSAM linear data set as a zFS aggregate. It supports both version 1.4 aggregates and version 1.5 aggregates.

Format

```
syscall_parmlist
   opcode              int           134       AGOP_FORMAT_PARMDATA
   parms[0]            int           offset to AGGR_ID
   parms[1]            int           offset to AGGR_FORMAT
   parms[2]            int           offset to system name (optional)
   parms[3]            int           0
   parms[4]            int           0
   parms[5]            int           0
   parms[6]            int           0
AGGR_ID
   aid_eye             char[4]       "AGID"
   aid_len             char          Sizeof(AGGR_ID)
   aid_ver             char          1
   aid_name            char[45]      Aggregate name
   aid_reserved        char[33]      0    (Reserved for the future)
AGGR_FORMAT
   af_eye              char[4]       "AGFM"
   af_len              short         Sizeof(AGGR_FORMAT)
   af_ver              char          1
   af_aggrversion      char          0 means honor format_aggrversion value
                                     4 means format a version 1.4 aggregate
                                     5 means format a version 1.5 aggregate
   af_size             int           Amount of aggregate to format
   af_logsize          int           Size of the aggregate log
   af_initialempty     int           this is ignored - always use 1
   af_overwrite        int           Use caution if you specify 1
   af_compat           int           Compat aggr desired (ignored;
                                     always compat)
   af_owner            int           No uid specified
   af_ownerSpecified   int           Use uid of issuer
   af_group            int           No guid specified
   af_groupSpecified   int           Gid set to issuer default group
   af_perms            int           No perms specified
   af_perms_specified  int           Perms not specified
   af_grow             int           Grow amount, 0 means grow not
                                     specified
   af_newauditfid      int           0=old auditfid; 1=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
  EINVAL        Invalid parameters
  EMVSERR       Internal error using an osi service
  ENOENT        No aggregate by this name is found
  EPERM         Permission denied to perform request

Reason_code
  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.
- The `af_compat` bit is ignored. The VSAM linear data set is always formatted as a compatibility mode aggregate.

Privilege required

The issuer must have ALTER authority on the VSAM linear data set to be formatted or must be logged in as root or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

Define Aggregate

Restrictions

The VSAM linear data set to be formatted cannot be attached.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

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
    char  af_aggrversion;            /* 0 means honor            */
                                     /* format_aggrversion value */
#define   AF_VERSION4 4
#define   AF_VERSION5 5
    int   af_size;                   /* Amount to format of aggr */
#define   AF_VERSION4 4              /* make a version 1.4 aggregate */
#define   AF_VERSION5 5              /* make a version 1.5 aggregate */
#define   AF_DEFAULT_SIZE 0          /* If set, we use default of entire */
                                     /* primary partition of LDS */
    int   af_logsize;                /* Size of logfile in aggr */
#define   AF_DEFAULT_LOGSIZE 0       /* If set, we use default of */
                                     /* 1% of aggr size           */
    int   af_initialempty;           /* Initial empty blocks */
#define   AF_DEFAULT_INITIALEMPTY 1  /* This is the default & minumum too */
    int   af_overwrite;              /* Overwrite aggr if its not empty */
#define   AF_OVERWRITE_OFF 0         /* 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_compat;                 /* HFS-compat aggr desired */
#define   AF_MULT 0                  /* HFS-compat aggr desired */
#define   AF_HFSCOMP 1               /* HFS-compat aggr desired */
    int   af_owner;                  /* Owner for HFS-compat */
    int   af_ownerSpecified;         /* Indicates an owner was provided */
#define   AF_OWNER_USECALLER 0       /* Owner is 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 */
    int   af_perms;                  /* Perms for HFS-compat */
#define   AF_DEFAULT_PERMS 0755      /* The default perms to use */
    int   af_permsSpecified;         /* Indicates if perms provided */
#define   AF_PERMS_DEFAULT 0         /* Perms not specified, use default */
#define   AF_PERMS_SPECIFIED 1       /* Use perms set in af_perms */
    int   af_grow;                   /* Amount to extend each time until */
                                     /* we reach desired size            */
                                     /* 0 means work the old way, just   */
                                     /* extend to desired size once      */
    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;

int main(int argc, char **argv)
{
    int    bpxrv;
    int    bpxrc;
    int    bpxrs;
    char   aggrname[45] = "PLEX.DCEIMGQX.LDS"; /* 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);*/

        myparmstruct.myparms.parms[3] = 0;
        myparmstruct.myparms.parms[4] = 0;
        myparmstruct.myparms.parms[5] = 0;
        myparmstruct.myparms.parms[6] = 0;

        memset(idp, 0, sizeof(AGGR_ID));
        memcpy(idp->aid_eye, AID_EYE, 4);
        idp->aid_ver = 1;
        strcpy(idp->aid_name, aggrname);
        idp->aid_len = (int)sizeof(AGGR_ID);
        memset(aggptr, 0, sizeof(myparmstruct.aggformat));
        memcpy(aggptr->af_eye, AF_EYE, 4);

        aggptr->af_len = sizeof(myparmstruct.aggformat);
        aggptr->af_ver = AF_VER_INITIAL;
        aggptr->af_size = AF_DEFAULT_SIZE;
        aggptr->af_compat = AF_HFSCOMP; /* HFS compatibility mode aggregate */

        /* aggptr->af_owner = owner; */
        aggptr->af_ownerSpecified = AF_OWNER_USECALLER;
        /* aggptr->af_group = group; */
        aggptr->af_groupSpecified = AF_GROUP_USECALLER;
        /* aggptr->af_perms = perms; */
        aggptr->af_permsSpecified = AF_PERMS_DEFAULT;

        aggptr->af_grow = 0;        /* no grow size */
        aggptr->af_aggrversion = 0; /* format with default version defined by */
                                    /* format_aggrversion value               */
        aggptr->af_newauditfid = 1; /* generate a new auditfid */

        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 formatting, BPXRV = %d BPXRC = %d BPXRS = %x\n",
                   bpxrv, bpxrc, bpxrs);
            return bpxrc;
        }
        else
            printf("Formatted aggregate %s\n", aggrname);

        return 0;
    }
```

Grow Aggregate

Purpose

Extends the physical size of an attached aggregate. It supports both version 1.4 aggregates and version 1.5 aggregates.

Format

```
syscall_parmlist
   opcode              int        129       AGOP_GROW_PARMDATA
   parms[0]            int        offset to AGGR_ID
   parms[1]            int        new size of aggregate
   parms[2]            int        0
   parms[3]            int        0
   parms[4]            int        0
   parms[5]            int        0
   parms[6]            int        0
AGGR_ID
   aid_eye             char[4]    "AGID"
   aid_len             char       sizeof(AGGR_ID)
   aid_ver             char       1   (new size is 32 bits)
   aid_name            char[45]   Name of aggregate
   aid_reserved        char[33]   0   ( Reserved for future use)

- OR -

syscall_parmlist
   opcode              int         129       AGOP_GROW_PARMDATA
   parms[0]            int         offset to AGGR_ID
   parms[1]            int         high 32 bits of new 64 bit size of aggregate
   parms[2]            int         low 32 bits of new 64 bit size of aggregate
   parms[3]            int         0
   parms[4]            int         0
   parms[5]            int         0
   parms[6]            int         0
AGGR_ID
   aid_eye             char[4]     "AGID"
   aid_len             char        sizeof(AGGR_ID)
   aid_ver             char        3  (new size is 64 bits)
   aid_name            char[45]    Name of aggregate
   aid_reserved        char[33]    0  (Reserved for future use)


Return_value 0 if request is successful, -1 if it is not successful

Return_code
  8        DFSMS did not extend the aggregate
  EBUSY    Aggregate is busy or otherwise unavailable
  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 notes

- The aggregate must be mounted or attached.
- The size specified is the new total size (in 1 KB blocks) that is being requested. The size can be rounded up by DFSMS. If a zero is specified for the new size,

the aggregate is grown by a secondary allocation. DFSMS determines whether to extend to another volume. Requests that write to files and need aggregate blocks that are not available yet and other requests that access those files will wait. Other requests will not wait during the grow.

- For an AGGR_ID version 1, the new size cannot be larger than approximately 4 TB. For an AGGR_ID version 3, the new size is a 64 bit number, and cannot be larger than approximately 16 TB.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must have ALTER authority on the VSAM linear data set to be formatted and must be logged in as root or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

List Aggregate Status Version 2

Restrictions

The aggregate to be grown cannot already be quiesced or be attached as read-only. An aggregate cannot be made smaller.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_GROW_PARMDATA 129

typedef struct syscall_parmlist_t {
    int  opcode;               /* Operation code to perform */
    int  parms[7];             /* Specific to type of operation, */
                               /* provides access to the parms */
                               /* parms[4]-parms[6] are currently unused*/
 } syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t {
    char aid_eye[4];                   /* Eye catcher */
#define AID_EYE "AGID"
    char aid_len;                      /* Length of this structure */
    char aid_ver;                      /* Version */
#define AID_VER_INITIAL 1              /* Initial version */
    char aid_name[ZFS_MAX_AGGRNAME+1]; /* Name, null terminated */
    char aid_reserved[33];             /* Reserved for the future */
} AGGR_ID;

struct parmstruct {
    syscall_parmlist myparms;
    AGGR_ID          aggr_id;
};

int main(int argc, char **argv)
{
    int             bpxrv;
    int             bpxrc;
    int             bpxrs;
    char            aggrname[45]     = "PLEX.DCEIMGQX.FS";

    struct parmstruct myparmstruct;

    /* Ensure reserved fields are 0 */
    memset(&myparmstruct.aggr_id, 0, sizeof(AGGR_ID));

    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 succssfully\n", aggrname);
    }
    return 0;
}
```

List Aggregate Status (Version 1)

Purpose

An aggregate operation that returns information about a specified attached aggregate on this system.

IBM recommends using the List Detailed File System Information API instead of List Aggregate Status or List File System Status.

Format

```
syscall_parmlist
   opcode              int             137       AGOP_GETSTATUS_PARMDATA
   parms[0]            int             offset to AGGR_ID
   parms[1]            int             offset to AGGR_STATUS
   parms[2]            int             0
   parms[3]            int             0
   parms[4]            int             0
   parms[5]            int             0
   parms[6]            int             0
AGGR_ID
   aid_eye             char[4]         "AGID"
   aid_len             char            sizeof(AGGR_ID)
   aid_ver             char            1
   aid_name            char[45]        "OMVS.PRV.AGGR001.LDS0001"
   aid_reserved        char[33]        0
AGGR_STATUS
   as_eye              char[4]         "AGST"
   as_len              short           sizeof(AGGR_STATUS)
   as_ver              char            1
   as_res1             char            0
   as_aggrId           int             Aggregate ID
   as_nFileSystems     int             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 int
   as_fragSize         int
   as_blockSize        int
   as_totalUsable      unsigned int
   as_realFree         unsigned int
   as_minFree          unsigned int
   as_reserved         char[128]

Return_value    0 if request is successful, -1 if it is not successful

Return_code
  EINTR         ZFS is shutting down
  EINVAL        Invalid parameter list
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
Reason_code
  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage notes

- To grow an aggregate, you need to specify a number larger than the sum of `as_totalUsable` and `as_minFree`.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

List Attached Aggregate Names

List Detailed File System Information

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_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 */
#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_status_t {
  char          as_eye[4];       /* Eye catcher */
#define AS_EYE "AGST"
  short         as_len;          /* Length of structure */
  char          as_ver;
#define AS_VER_INITIAL 1         /* Initial version */
  char          as_res1;         /* Reserved. */
  int           as_aggrId;       /* Internal identifier */
  int           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 */
    unsigned int  as_blocks;        /* Number of fragments in aggregate */
    int           as_fragSize;      /* Size of fragment in
                                       aggregate (normally 1K) */
    int           as_blockSize;     /* Size of blocks on
                                       aggregate (normally 8K) */
    unsigned int  as_totalUsable;   /* Total available blocks on
                                       aggregate (normally 8K) */
    unsigned int  as_realFree;      /* Total kilobytes free */
    unsigned int  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;

    /* aggregate name to getstatus */
    char              aggrname[45] = "PLEX.DCEIMGQX.FS";
    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

Returns information about a specified attached aggregate on this system. Version 2 returns additional flags and fields.

| IBM recommends that you should use the List Detailed File System Information
| API instead of List Aggregate Status or List File System Status.

Format

```
| syscall_parmlist
|    opcode                 int               146   AGOP_GETSTATUS2_PARMDATA
|    parms[0]               int               Offset to AGGR_ID
|    parms[1]               int               Offset to AGGR_STATUS2
|    parms[2]               int               0
|    parms[3]               int               0
|    parms[4]               int               0
|    parms[5]               int               0
|    parms[6]               int               0
| AGGR_ID
|    aid_eye                char[4]           "AGID"
|    aid_len                char              Sizeof(AGGR_ID)
|    aid_ver                char              1
|    aid_name               char[45]          Aggregate name
|    aid_reserved           char[33]          0
```

```
AGGR_STATUS2
   as_eye                     char[4]           "AGST"
   as_len                     short             Sizeof(AGGR_STATUS2)
   as_ver                     char              2
   as_res1                    char              0
   as_aggrId                  int               Aggregate ID
   as_nFileSystems            int               Number of File Systems
   as_threshold               char              Aggrfull threshold
   as_increment               char              Aggrfull increment
   as_flags                   char
     AS_MONITOR                                 0x80 Monitoring for aggrfull
     AS_RO                                      0x40 Attached Read-only
     AS_NBS                                     0x20 NBS being guaranteed
     AS_COMPAT                                  0x10 Formatted as HFS-compat
     AS_GROW                                    0x08 Can be dynamically grown
     AS_QUIESCED                                0x01 1 means aggr is quiesced
   as_flags2                  char
     AS_DISABLED                                0x80 Aggr is disabled
     AS_SYSPLEXAWARE                            0x40 Aggr mounted RWSHARE and
                                                     is sysplex-aware
   as_blocks                  unsigned int      Number of 8K blocks in aggr
   as_fragSize                int               Number of 1K fragments in aggr
   as_blockSize               int               Size of blocks (8K normally)
   as_totalUsable             unsigned int      Total available blocks
   as_realFree                unsigned int      Total free 1K blocks
   as_minFree                 unsigned int      Minimum kilobytes free
   as_reserved2               int[3]            Reserved
   as_freeblocks              unsigned int      K available in free 8K blocks
   as_freefrags               unsigned int      K available in free 1K frags
   as_directLog               unsigned int      K used on the log
   as_indirectLog             unsigned int      K used indirectly on the log
   as_fstbl                   unsigned int      K used for file system table
   as_bitmap                  unsigned int      K used for the bitmap
   as_diskFormatMajorVersion  unsigned int      Disk format major version
   as_diskFormatMinorVersion  unsigned int      Disk format minor version
```

```
s_auditfid                char[10]          Aggregate Audit Fid
   as_bytes_reserved          char[2]           Reserved
   as_reserved3               int               Reserved
   as_quiesce_time            struct timeval    If quiesced, time quiesce
                                                occurred
      posix_time_low          int               Seconds since epoch
      posix_usecs             int               Micro-seconds
   as_quiesce_jbname          char[9]           If quiesced, Job name
                                                requesting quiesce
   as_quiesce_sysname         char[9]           If quiesced, system name
                                                quiesce request came from
   as_reserved                char[42]          Reserved

OR

  syscall_parmlist
   opcode                     int               146   AGOP_GETSTATUS2_PARMDATA
   parms[0]                   int               Offset to AGGR_ID
   parms[1]                   int               Offset to AGGR_STATUS3
   parms[2]                   int               0
   parms[3]                   int               0
   parms[4]                   int               0
   parms[5]                   int               0
   parms[6]                   int               0
AGGR_ID
   aid_eye                    char[4]           "AGID"
   aid_len                    char              Sizeof(AGGR_ID)
   aid_ver                    char              1
   aid_name                   char[45]          Aggregate name
   aid_reserved               char[33]          0
AGGR_STATUS3
   as_eye                     char[4]           "AGST"
   as_len                     short             sizeof(AGGR_STATUS2)
   as_ver                     char              3  (supports 64 bit sizes)
   as_res1                    char              0
   as_aggrId                  int               Aggregate ID
   as_nFileSystems            int               Number of File Systems
   as_threshold               char              Aggrfull threshold
   as_increment               char              Aggrfull increment
   as_flags                   char
     AS_MONITOR                                 0x80 Monitoring for aggrfull
     AS_RO                                      0x40 Attached Read-only
     AS_NBS                                     0x20 NBS being guaranteed
     AS_COMPAT                                  0x10 Formatted as HFS-compat
     AS_GROW                                    0x08 Can be dynamically grown
     AS_QUIESCED                                0x01 1 means aggr is quiesced
   as_flags2                  char
     AS_DISABLED                                0x80 Aggr is disabled
     AS_SYSPLEXAWARE                            0x40 Aggr mounted RWSHARE and
                                                     is sysplex-aware
     AS_CONVERTTOV5                             0x20 Aggregate enabled for
                                                     automatic V5 conversion
   as_blocks                  unsigned int      Number of 8K blocks in aggr
   as_fragSize                int               Number of 1K fragments in aggr
   as_blockSize               int               Size of blocks (8K normally)
   as_totalUsable             unsigned int      Total available blocks
   as_realFree                unsigned int      Total free 1K blocks
   as_minFree                 unsigned int      Minimum kilobytes free
   as_reserved2               int[3]            Reserved
   as_freeblocks              unsigned int      K available in free 8K blocks
   as_freefrags               unsigned int      K available in free 1K frags
   as_directLog               unsigned int      K used on the log
   as_indirectLog             unsigned int      K used indirectly on the log
   as_fstbl                   unsigned int      K used for file system table
   as_bitmap                  unsigned int       K used for the bitmap
   as_diskFormatMajorVersion  unsigned int       Disk format major version
   as_diskFormatMinorVersion  unsigned int       Disk format minor version
   as_auditfid                char[10]           Aggregate Audit Fid
   as_bytes_reserved          char[2]            Reserved
   as_reserved3               int                Reserved

   as_quiesce_time            struct timeval     If quiesced, time quiesce
                                                 occurred
      posix_time_low          int                Seconds since epoch
      posix_usecs             int                Micro-seconds
   as_quiesce_jbname          char[9]            If quiesced, Job name
                                                 requesting quiesce
   as_quiesce_sysname         char[9]            If quiesced, system name
                                                 quiesce request came from
   as_pad                     char[6]            Gets alignment
   as_blocks_hyper            hyper              Number of 8K blocks in aggr
   as_totalUsable_hyper       hyper              Total available blocks
   as_realFree_hyper          hyper              Total free 1K blocks
   as_minFree_hyper           hyper              Minimum kilobytes free
   as_freeblocks_hyper        hyper              K available in free 8K blocks
   as_freefrags_hyper         hyper              K available in free 1K frags
   as_directLog_hyper         hyper              K used on the log
   as_indirectLog_hyper       hyper              K used indirectly on the log
   as_fstbl_hyper             hyper              K used for file system table
   as_bitmap_hyper            hyper              K used for the bitmap
   as_reserved                char[42]           Reserved for future use

Return_value 0 if request is successful, -1 if it is not successful
Return_code
  EINTR         ZFS is shutting down
  EINVAL        Invalid parameter list
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
Reason_code
  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage notes

- The aggregate must be mounted or attached.
- To grow an aggregate, you need to specify a number larger than the sum of `as_totalUsable` and `as_minFree`.
- For an AGGR_STATUS2, if a size is too large for 32 bits, 0xFFFFFFFF is returned. For an AGGR_STATUS3, sizes are returned in both the normal fields and the hyper fields.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

List Attached Aggregate Names
List Detailed File System Information

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_GETSTATUS2_PARMDATA 146

typedef struct syscall_parmlist_t {
  int           opcode;       /* Operation code to perform */
  int           parms[7];     /* Specific to type of operation, */
                              /* provides access to the parms */
                              /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct timeval {
  int           posix_time_low; /* seconds since epoch */
  int           posix_usecs;    /* microseconds */
} TIMEVAL;

#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t {
  char          aid_eye[4];                 /* Eye Catcher */
#define AID_EYE "AGID"
  char          aid_len;                    /* Length of this structure */
  char          aid_ver;                    /* Version */
#define AID_VER_INITIAL 1                   /* Initial version */
  char          aid_name[ZFS_MAX_AGGRNAME+1]; /* aggr name, null terminated */
  char          aid_reserved[33];           /* Reserved for the future */
} AGGR_ID;

typedef struct aggr_status_t {
  char          as_eye[4];          /* Eye catcher */
#define AS_EYE "AGST"
  short         as_len;             /* Length of structure */
  char          as_ver;
#define AS_VER_2 2                  /* version 2 */
  char          as_res1;            /* Reserved. */
  int           as_aggrId;          /* Internal identifier */
  int           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 */
#define AS_SYSPLEXAWARE 0x40                /* Aggr is sysplex-aware
                                               for r/w. Attached but not
                                               mounted compats will never
                                               have AS_SYSPLEXAWARE on */

  unsigned int  as_blocks;      /* Number of fragments in aggregate */
  int            as_fragSize;    /* Size of fragment in aggregate
                                    (normally 1K) */
  int            as_blockSize;   /* Size of blocks on aggregate (normally 8K)*/
  unsigned int   as_totalUsable; /* Total available blocks on aggregate
                                    (normally 8K) */
  unsigned int   as_realFree;    /* Total kilobytes free */
  unsigned int   as_minFree;     /* Minimum kilobytes free */
  int            as_reserved2[3];
  unsigned int   as_freeblocks;  /*Number of k available in free 8k blocks*/
  unsigned int   as_freefrags;   /*Number of k available in free 1k fragments*/
  unsigned int   as_directLog;   /*Number of k used on the log*/
  unsigned int   as_indirectLog; /*Number of k used indirectly on the log*/
  unsigned int   as_fstbl;       /*Number of k used for the filesystem table*/
  unsigned int   as_bitmap;      /*Number of k used for the bitmap file*/
  unsigned int   as_diskFormatMajorVersion;   /* disk format major version */
  unsigned 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 */
  int            as_reserved3;
  struct timeval as_quiesce_time;             /* time of last quiesce */
  char           as_quiesce_jbname[9];        /* job name of last quiesce -
                                                 null terminated */
  char           as_quiesce_sysname[9];       /* system where last quiesce
                                                 issued - null terminated */
  char           as_reserved[42];             /* Reserved for future */
} AGGR_STATUS2;

struct parmstruct {
  syscall_parmlist myparms;
  AGGR_ID          aggr_id;
  AGGR_STATUS2     aggr_status;
};

int main(int argc, char **argv)
{
  int               bpxrv;
  int               bpxrc;
  int               bpxrs;
  int               i;
  char              buf[33];
  unsigned int      ptl;

  /* aggregate name to getstatus */
  char              aggrname[45] = "PLEX.DCEIMGQX.FS";

  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.%d\n",
         myparmstruct.aggr_status.as_diskFormatMajorVersion,
         myparmstruct.aggr_status.as_diskFormatMinorVersion);

  printf("getstatus: auditfid=");

  for (i = 0; i < 10; i++)
    printf("%2.2X", myparmstruct.aggr_status.as_auditfid[i]);

  printf("\n");

  if (myparmstruct.aggr_status.as_flags & AS_QUIESCED)
  {
    if (myparmstruct.aggr_status.as_quiesce_jbname[0] != 0x00)
    {
        ptl = myparmstruct.aggr_status.as_quiesce_time.posix_time_low;
        if (0 == ctime_r((time_t * ) & ptl, 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",
                  myparmstruct.aggr_status.as_quiesce_time.posix_usecs);
          buf[26] = ' ';
          buf[19] = '.';
          printf("Quiesced by job %s on system %s on %s",
                 myparmstruct.aggr_status.as_quiesce_jbname,
                 myparmstruct.aggr_status.as_quiesce_sysname,
                 buf);
        }
      }
    }
    printf("\n");
  }
  return 0;
}
```

List Attached Aggregate Names (Version 1)

Purpose

List Attached Aggregate Names (Version 1) is an aggregate operation that returns a list of the names of all attached aggregates on a system.

Format

```
syscall_parmlist
   opcode          int            135       AGOP_LISTAGGRNAMES_PARMDATA
   parms[0]        int            buffer length or 0
   parms[1]        int            offset to AGGR_ID or 0
   parms[2]        int            offset to size
   parms[3]        int            offset to system name (optional)
   parms[4]        int            0
   parms[5]        int            0
   parms[6]        int            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        int            bytes returned or size needed
                                  if the return code is E2BIG

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 notes

- This call returns an array of AGGR_IDs, one for each attached aggregate on the system. Each AGGR_ID structure is 84 bytes. You can specify a buffer that you think might hold all of them or you can specify a buffer length and offset to AGGR_ID 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

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#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 */
#define  AID_EYE "AGID"
  char aid_len;                      /* Length of this structure */
  char aid_ver;                      /* Version */
#define  AID_VER_INITIAL 1           /* Initial version */
  char aid_name[ZFS_MAX_AGGRNAME+1]; /* aggr name, null terminated */
  char aid_reserved[33];             /* Reserved for the future */
} AGGR_ID;

struct parmstruct {
  syscall_parmlist myparms;
  /* Real malloc'd structure will have an array of AGGR_IDs here */
  int              size;
  char             systemname[9];
};

int main(int argc, char **argv)
{
  int               bpxrv;
  int               bpxrc;
  int               bpxrs;
  struct parmstruct myparmstruct;
  AGGR_ID           *aggPtr;
  int               aggSize      = sizeof(AGGR_ID);
  int               buflen       = sizeof(AGGR_ID);
  struct parmstruct *myp         = &myparmstruct;
  int               mypsize;
  char              *systemp;
  int               count_aggrs,
    total_aggrs;

  myparmstruct.myparms.opcode   = AGOP_LISTAGGRNAMES_PARMDATA;
  myparmstruct.myparms.parms[0] = 0;
  myparmstruct.myparms.parms[1] = 0;
  myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist);
  myparmstruct.myparms.parms[3] = 0;
  myparmstruct.myparms.parms[4] = 0;
  myparmstruct.myparms.parms[5] = 0;
  myparmstruct.myparms.parms[6] = 0;

  BPX1PCT("ZFS     ",
          ZFSCALL_AGGR,              /* Aggregate operation */
```

```
                sizeof(myparmstruct),       /* Length of Argument */
                (char *)&myparmstruct,      /* Pointer to Argument */
                &bpxrv,                     /* Pointer to Return_value */
                &bpxrc,                     /* Pointer to Return_code */
                &bpxrs);                    /* Pointer to Reason_code */

        if (bpxrv < 0)
        {
          if (bpxrc == E2BIG)
          {
            buflen = myp->size;         /* Get buffer size needed */
            mypsize = buflen + sizeof(syscall_parmlist) + sizeof(int) + 9;
            myp = (struct parmstruct *)malloc((int)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(int);                                             */
            /* 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(int);                              */

            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 */

            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);
            }
            else
            {   /* lsaggr names failed with large enough buffer */
              printf("Error on ls aggr with large enough buffer\n");
              printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
              free(myp);
              return bpxrc;
            }
          }
```

```
      else
      {   /* error was not E2BIG */
        printf("Error on ls aggr trying to get required size\n");
        printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
        free(myp);
        return bpxrc;
      }
    }
    else
    {   /* asking for buffer size gave rv = 0; maybe there are no aggregates */
      if (myparmstruct.size == 0)
        printf("No attached aggregates\n");
      else  /* No, there was some other problem with getting the size needed */
        printf("Error getting size required\n");
    }
    return 0;
  }
```

List Attached Aggregate Names (Version 2)

Purpose

The List Attached Aggregate Names (Version 2) subcommand call returns a list of the names of all attached aggregates on a system with the system name.

Format

```
syscall_parmlist
   opcode          int           140       AGOP_LISTAGGRNAMES2_PARMDATA
   parms[0]        int           buffer length or 0
   parms[1]        int           offset to AGGR_ID2 or 0
   parms[2]        int           offset to size
   parms[3]        int           offset to system name (optional)
   parms[4]        int           0
   parms[5]        int           0
   parms[6]        int           0
AGGR_ID2[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               int           bytes returned or size needed
                                 if the return code is E2BIG
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 notes

- This call returns an array of AGGR_ID2 structures, one for each attached aggregate on the system. Each AGGR_ID2 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 to AGGR_ID2 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

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

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 */
#define  AID_EYE "AGID"
  char aid_len;                        /* Length of this structure */
  char aid_ver;                        /* Version */
#define  AID_VER_2 2                     /* version 2 */
  char aid_name[ZFS_MAX_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 */
  int            size;
  char           systemname[9];
};

int main(int argc, char **argv)
{
  int              buffer_success = 0;
  int              bpxrv;
  int              bpxrc;
  int              bpxrs;
  int              t;
  struct parmstruct myparmstruct;
  AGGR_ID2         *aggPtr;
  int              aggSize      = sizeof(AGGR_ID2);
  int              buflen       = sizeof(AGGR_ID2);
  struct parmstruct *myp        = &myparmstruct;
  int              mypsize;
  char             *systemp;
  int              count_aggrs;
  int              total_aggrs;

  myparmstruct.myparms.opcode  = AGOP_LISTAGGRNAMES2_PARMDATA;
  myparmstruct.myparms.parms[0] = 0;
  myparmstruct.myparms.parms[1] = 0;
  myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist);
  myparmstruct.myparms.parms[3] = 0;
  myparmstruct.myparms.parms[4] = 0;
  myparmstruct.myparms.parms[5] = 0;
  myparmstruct.myparms.parms[6] = 0;
```

```
      BPX1PCT("ZFS     ",
              ZFSCALL_AGGR,                /* Aggregate operation */
              sizeof(myparmstruct),        /* Length of Argument */
              (char *)&myparmstruct,       /* Pointer to Argument */
              &bpxrv,                      /* Pointer to Return_value */
              &bpxrc,                      /* Pointer to Return_code */
              &bpxrs);                     /* Pointer to Reason_code */

      for(t = 0; t < 1000 && buffer_success == 0; t++)
      {
        if (bpxrv < 0)
        {
          if (bpxrc == E2BIG)
          {
            buflen = myp->size;          /* Get buffer size needed */
            mypsize = buflen + sizeof(syscall_parmlist) + sizeof(int) + 9;

            free(myp);

            myp = (struct parmstruct *)malloc((int)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(int);        */
            /* 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(int);     */

            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 */

            if( bpxrv != 0 && bpxrc == E2BIG )
              printf("E2BIG: %d times total\n", t++);
            else if( bpxrv == 0 )
            {
              buffer_success = 1;
              total_aggrs = buflen / aggSize;
              count_aggrs = 1;
              for (aggPtr = (AGGR_ID2 * ) & (myp->size);
                   count_aggrs <= total_aggrs;
                   aggPtr++, count_aggrs++)
              {
                if (strlen(aggPtr->aid_name) != 0)
                  printf("%-64.64s %-8.8s\n",
                         aggPtr->aid_name, aggPtr->aid_sysname);
              }
              free(myp);
            }
            else
            { /* lsaggr names failed with large enough buffer */
              printf("Error on ls aggr with large enough buffer\n");
              printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
              free(myp);
              return bpxrc;
            }
          }
          else
          { /* error was not E2BIG */
            printf("Error on ls aggr trying to get required size\n");
            printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
            free(myp);
            return bpxrc;
          }
        }
        else
        { /* asking for buffer size gave rv = 0; maybe there are no aggregates */
          if (myparmstruct.size == 0)
            printf("No attached aggregates\n");
          else /* No, there was some other problem with getting the size needed */
            printf("Error getting size required\n");
          free(myp);
          return bpxrc;
        }
      }

      if( t == 1000 )
        printf("Number of failed buffer resizes exceeded.\n");

      free(myp);
      return 0;
    }
```

List Detailed File System Information

Purpose

Returns detailed information for one or more file systems. It offers the ability to get information for file systems that have common names, common attributes, or that have encountered similar unexpected conditions.

IBM recommends that you use the List Detailed File System Information API instead of List Aggregate Status, List File System Status, List File System Names (Version 1) or List File System Names (Version 2).

Format

```
syscall_parmlist
  opcode            int            153    AGOP_FSINFO_PARMDATA
                                   154    AGOP_FSINFO_RESET_PARMDATA
  parms[0]          int            offset to FSINFO_REQUEST
  parms[1]          int            0
  parms[2]          int            0
  parms[3]          int            0
  parms[4]          int            0
  parms[5]          int            0
  parms[6]          int            0

FSINFO_REQUEST
  fr_eye            char[4]        "FIRQ"
  fr_length         short          Length of Structure
  fr_sversion       char           Structure Version, must be 1
  fr_reqtype        char           SingleQuery=0, NameCursor=1
  fr_version        char           Version of input/output
                                    buffer, must be 1
  fr_output         char           Type of output/function selected, one of:
                                    0 - Local statistics only, use only local
                                        cache. Only allowed with
                                        fr_nameSelection=2.
                                    1 - Full sysplex-wide statistics
                                        (including owner statistics).
                                    2 - Reset statistics.
  fr_nameSelection  char           Selection of aggregates desired, one of:
                                    0 - When SingleQuery selected.
                                   Options for fr_reqtype=1 (NameCursor):
                                    1 - All aggregates. fr_output can be
                                        1 (full) or 2 (reset).
                                    2 - Aggregates known on the local system.
                                        This is only allowed with fr_output
                                        0 (local statistics).
                                    3 - All aggregates matching a specific
                                        pattern provided in fr_patternName.
                                        fr_output can be 1 (full) or
                                        2 (reset).
```

```
fr_eol           char           Indicates if a multi-aggregate read has
                                 completed. 1 if yes, 0 if no.
fr_selection     int            Selection mask for aggregates meeting
                                 certain state criteria. More than one bit
                                 can be set. Note zFS will use an OR-ing
                                 of the criteria so that aggregates that
                                 meet one or more criteria are returned.
                                 0 - all aggregates desired.
                                 x1 - Show aggregates that have sysplex
                                      thrashing objects.
                                 x2 - Show aggregates that contain v5
                                      directories with overflow pages.
                                 x4 - Show aggregates mounted R/W.
                                 x8 - Show aggregates mounted R/O.
                                 x10 - Show aggregates that are disabled.
                                 x20 - Show aggregates that are growing.
                                 x40 - Show aggregates that are quiesced.
                                 x80 - Show aggregates that had grow
                                       failures.
                                 x100 - Show aggregates that are low on
                                        space, as defined by the zFS
                                        bitmap manager.
                                 x200 - Show aggregates that are damaged.
                                 x400 - Show aggregates that are mounted
                                        RWSHARE.
                                 x800 - Show aggregates that are mounted
                                        NORWSHARE.
                                 x1000 - Show aggregates that had requests
                                 x2000 - Show aggregates that had write
                                         requests.
                                 x4000 - Show aggregates where applications
                                         saw ENOSPC errors.
                                 x8000 - Show aggregates that had disk I/O
                                         errors.
                                 x10000 - Show aggregates that had XCF
                                          timeouts between client systems
                                          and owning systems (for RWSHARE
                                          aggregates).
                                 x20000 - Show aggregates that are version
                                          1.4 aggregates.
                                 x40000 - Show aggregates that are version
                                          1.5 aggregates.
                                 x80000 - Show aggregates that are
                                          disabled for dynamic grow.
                                 x100000 - Show aggregates that are
                                           disabled for conversion to
                                           version 1.5.
                                 x80000000 - Tells zFS to use an AND-ing
                                             method of examining criteria.
                                             Hence only aggregates meeting
                                             all criteria will be returned.
                                 x801FFFFF - Represents all valid bits.
fr_entries       unsigned int   Number of aggregatess returned in output.
fr_nonFatalRc    int            Non-fatal error code.
fr_nonFatalRsn   int            Reason code if fr_nonFatalRc is non-zero.
fr_resumeName    char[45]       Dataset name to resume with for NameCursor
                                 or the name of a single-aggregate query.
fr_patternName   char[45]       The aggregate name to be used. This can
                                 contain wildcards.
fr_future2       char[2]        For future use (reserved).
```

```
FSINFO_NAME
  fn_eye            char[4]         "FINA"
  fn_slength        short           Structure length.
  fn_sversion       short           Structure version, must be 1.
  fn_name           char[44]        Aggregate name.
  fn_connected      unsigned int    Number of connected systems if owner
                                     output is requested; 0 otherwise.
  fn_owner          char[8]         System name of the owner.
  fn_length         unsigned int    Total length of all information for this
                                    aggregate.
  fn_future         char[4]         For future use (reserved).
  fn_sysnames       char[8]         Names of connected systems (32 at most).

FSINFO_OWNER
  fo_eye            char[4]         "FIOW"
  fo_length         short           Length of structure
  fo_sversion       short           Structure version, must be 1.
  fo_size           unsigned int    Number of 8K blocks in the aggregate.
  fo_free           unsigned int    Number of unused 8K blocks in the
                                     aggregate.
  fo_frags          unsigned long long int  Number of free 1K fragments
                                             available in the aggregate.
  fo_logsize        unsigned int    Number of 8K blocks allocated to the log
                                     file for transaction logging, including
                                     indirect blocks.
  fo_bitmapsize     unsigned int    Number of 8K blocks allocated to the
                                     bitmap file, including indirect blocks.
  fo_anodesize      unsigned int    Number of 8K blocks allocated to the
                                     anode table.
  fo_objects        unsigned int    Number of objects in the file system.
  fo_version        char            Aggregate version number.
  fo_threshold      char            Space monitoring threshold.
  fo_increment      char            Space monitoring increment.
  reserved1         char            Reserved
  fo_flags          int             Flag bits:
                                     x01 - Mounted in R/W mode.
                                     x02 - Disabled for access.
                                     x04 - Grow failure occurred since
                                            last reset.
                                     x08 - Aggregate is low on space
                                            (zfs definition).
                                     x10 - Aggregate considered damaged by
                                            salvage verification and not
                                            repaired yet.
                                     x20 - Aggregage using zFS sysplex sharing
                                            (RWSHARE).
                                     x40 - Dynamic grow set at mount time.
                                     x80 - Aggregate is in the process of
                                            growing at time of query.
                                     x100 - converttov5 is set.
                                     x200 - Aggregate is not mounted.
                                     x400 - Aggregate is unowned.
                                     x800 - Dynamic grow allowed, no grow
                                             failures or since a grow failure
                                             an admin grow was done.
                                     x1000 - The quiesce is done for chgowner.
                                     x2000 - converttov5 disabled.
                                     x4000 - Aggregate version 1.4.
                                     x8000 - Aggregate version 1.5.
  fo_overflow        unsigned int    Number of overflow pages used in v5
                                      directories.
  fo_overflowhiwater unsigned int    Hi-water mark of fo_overflow for life of
                                      the file system.
  fo_thrashing       unsigned int    Current number of objects using the
                                      thrash-resolution protocol.
  reserved2          char[4]         Reserved.
  fo_thrash_resolution unsigned long long int  Number of thrash resolutions
                                                performed since last statistics
                                                reset.
  fo_revocations     unsigned long long int  Number of token revocations
                                              performed since last statistics
                                              reset.
  fo_revwait         unsigned long long int  Average revocation wait time in
                                              microseconds.
  fo_qsysname        char[8]         Name of system requesting quiesce, if the
                                      aggregate is quiesced, 0 otherwise.
  fo_jobname         char[8]         Name of job requesting the quiesce, if the
                                      aggregate is quiesced, 0 otherwise.
  fo_createtime      unsigned long long int  Creation time in seconds since
                                              last epoch.
  fo_ownership       unsigned long long int  Owership time in seconds since
                                              last epoch.
  fo_reset           unsigned long long int  Time statistic counters reset
                                              in seconds since last epoch.
  fo_quiesce         unsigned long long int  Quiesce time in seconds since
                                              epoch, 0 if not quiesced.
  fo_devno           unsigned int    z/OS UNIX device number.
  fo_auditfid        char[10]        Audit fid for file system.
  fo_qasid           unsigned short  ASID which issued the quiesce.
  fo_growcount       unsigned int    Number of grows since mount.
  reserved3          char[4]         Reserved.
  fo_growtime        unsigned long long int  Time of the last grow as known
                                              by the owner.

FSINFO_LOCAL
  fl_eye             char[4]                 "FILO"
  fl_length          short                   Structure Length.
  fl_sversion        short                   Structure version.
  fl_vnodes          unsigned long long int  Number of vnodes cached in
                                              memory on the local system.
  fl_ussheld         unsigned long long int  Number of vnodes held by USS.
  fl_sysname         char[8]                 System name stats are for.
  fl_open            unsigned long long int  Number of open objects in the
                                              file system.
  fl_tokens          unsigned long long int  Number of tokens held from the
                                              token manager.
  fl_usercache       unsigned int            Number of 4K pages held in the
                                              user cache for the file system.
  fl_metacache       unsigned int            Number of 8K pages held in the
                                              metadata cache.
  fl_appreads        unsigned long long int  Number of application reads
                                              done since last reset.
  fl_appreadresp     unsigned long long int  Average read response time,
                                              in microseconds.
  fl_appwrites       unsigned long long int  Number of application writes
                                              done since last reset.
  fl_appwriteresp    unsigned long long int  Average write response time,
                                              in microseconds.
  fl_xcfreads        unsigned long long int  Number of XCF read calls made
                                              to the owner since last reset.
  fl_xcfreadresp     unsigned long long int  Average XCF read call response
                                              time, in microseconds.
  fl_xcfwrites       unsigned long long int  Number of XCF write calls made
                                              to the server since last reset.
  fl_xcfwriteresp    unsigned long long int  Average XCF write call response
                                              time, in microseconds.
  fl_enospc          unsigned long long int  Number of ENOSPC errors
                                              returned to applications
                                              since last reset.
  fl_ioerrs          unsigned long long int  Number of disk I/O errors
                                              since last reset.
  fl_commerrs        unsigned long long int  Number of XCF communication
                                              timeouts or failures since last
                                              reset.
  fl_cancels         unsigned long long int  Number of cancelled operations
                                              since last reset by asynchronus
                                              abends, cancels, or forces.
  fl_ddname          char[8]                 DDNAME during allocation of
                                              aggregagte dataset.
  fl_mounttime       struct timeval64        Mount time in seconds since
                                              the last epoch.
  fl_numdasd         unsigned int            Number of DASD volumes listed
                                              for aggregate in FSINFO_DASD
                                              array.
  fl_flags           unsigned int            1 indicates this system has tasks
                                              waiting on a quiesced FS.

FSINFO_DASD
  fd_eye             char[4]                 "FIDA"
  fd_length          short                   Structure Length.
  fd_sversion        short                   Structure version, must be 1.
  fd_volser          char[6]                 Volume serial.
  fd_pavios          short                   Number of I/Os zFS will issue
                                              at one time for non-critical
                                              I/Os.
  fd_reads           unsigned long long int  Number of reads to this volume.
  fd_readbytes       unsigned long long int  Number of kilobytes read.
  fd_writes          unsigned long long int  Number of writes to this volume
  fd_writebytes      unsigned long long int  Number of kilobytes written.
  fd_waits           unsigned long long int  Number of times a zFS task had
                                              to wait for an I/O to this
                                              volume.
  fd_waitTime        unsigned long long int  (includes all time, queue wait,
                                             DASD response time etc.)
                                              since last reset.
  fd_resptime        unsigned long long int  Avg. wait time in microseconds.

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
  ENOENT          Specified data set is not found
  EPERM           Permission denied to perform request
Reason_code
  0xEFnnxxxx      See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Users of the API supply an input buffer that contains a `syscall_parmlist` followed by an FSINFO_REQUEST structure. Output will be placed in this buffer after the FSINFO_REQUEST.
- A minimum length output buffer for a single-aggregate query is 10 K, and a minimum length output buffer for a multi-aggregate query is 64 K.
- A single specific aggregate can be queried by putting its name in `fr_resumeName`. The name must be null-terminated. Also specify `fr_reqtype 0 (SingleQuery)`. This aggregate does not need to be attached. `fr_selection` and `fr_nameSelection` must also be 0.
- Multiple aggregate names can be specified by entering a string in `fr_patternName` that can contain a wildcard character ('`*`'). A wildcard can be specified at the beginning, at the end, or both at the beginning and the end of the string. The string must be null-terminated. The input string is converted to uppercase before it is processed. Use a `fr_nameSelection` value of 3 when specifying a wildcard, and a `fr_reqtype` of NameCursor (1).
- All attached aggregates can be specified by using `fr_nameSelection` value of 1 and a `fr_reqtype` value of NameCursor (1).
- If the output buffer cannot hold all of the returned information, `fr_eol` will be 0 and `fr_resumeName` will contain a value to be returned to zFS on the next query. Keep querying zFS until `fr_eol` is 1 to indicate that all information has been returned.
- Use `fr_selection` to return only aggregates that match the specified criteria in a multiple aggregate query. The options are defined in the Format section.
- `fr_output` determines the output of the request. Options are defined in the Format section.
- There is no file system information returned when a reset is requested (`fr_output=2`). A reset can only be requested when the opcode is 154 (AGOP_FSINFO_RESET_PARMDATA) and `fr_selection` is 0.
- Reserved fields and undefined flags must be set to binary zeros.
- Any names returned that are less than the full length of the field are null terminated. If the length of the name is equal to the length of the field that contains it, then it is not null terminated.
- Output consists of various structures following the FSINFO_REQUEST area in the buffer. For each aggregate that has information returned, first will be an FSINFO_NAME structure. This contains the name of an aggregate and the systems that are connected to it. Then, if present, will be the FSINFO_OWNER structure. This contains aggregate statistics and attributes as known by the owner. There can be no FSINFO_OWNER in some cases when the aggregate is unowned (`fn_owner` is *UNOWNED). This is followed by FSINFO_LOCAL structures. There are fn_connected FSINFO_LOCAL structures (if it is unowned), otherwise there are fn_connected+1 FSINFO_LOCAL structures. Each FSINFO_LOCAL structure is followed by `fl_numdasd` FSINFO_DASD structures to describe the DASD volumes that contain the zFS aggregate data set.

Privilege required

If a reset of the statistics values is requested and the `fr_output` field of the FSINFO_REQUEST structure contains the value 2, the issuer must be UID 0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class. Otherwise, no privilege is required.

Related services

List Aggregate Status (Version 1)
List Aggregate Status (Version 2)
List Attached Aggregate Names (Version 1)
List Attached Aggregate Names (Version 2)
List File System Names (Version 1)
List File System Names (Version 2)
List File System Status

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
#include <stddef.h>
#include <stdint.h>
#include <time.h>

#define ZFSCALL_FSINFO       0x40000013
#define ZFS_MAX_AGGRNAME     44
#define AGOP_FSINFO_PARMDATA 153 /* Get status on aggr & fs */
#define BUFFER_SIZE          1024 * 64

#define FSINFO_XCF_ERR       0x1
#define FSINFO_IO_ERR        0x2
#define FSINFO_SPC_ERR       0x4

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[1]-parms[6] are currently unused*/
} syscall_parmlist;

struct timeval64 {
  uint64_t tv_sec;
  int32_t  tv_usec_pad;
  uint32_t tv_usec;
};

typedef struct FSINFO_REQUEST_t {
  char  fr_eye[4];
#define FR_EYE  "FIRQ"
  short fr_length;
  char  fr_sversion;             /* Structure version. must be 1 */
  char  fr_reqtype;              /*request type. BulkList=0, OffsetCursor=1*/
#define FR_REQTYPE_SINGLEQUERY 0
#define FR_REQTYPE_NAMECURSOR  1
  char  fr_version;              /* Version of input/output buffer; must be 1*/
  char  fr_output;               /* Type of output */
#define FR_OUT_LOCAL_STAT      0 /* Local stats from local system */
#define FR_OUT_FULL_STAT       1 /* Full stats from all systems*/
#define FR_OUT_RESET           2 /* reset statistics */
  char  fr_nameSelection;        /* Selection of aggregates desired, one of: */
#define FR_NM_ALLAGGR          1 /* All aggregates */
#define FR_NM_LOCAL            2 /* Local aggregates */
#define FR_NM_PATTERN          3 /* All aggregates matching pattern */
  char  fr_eol;                  /* Indicates if a multi-aggregate
```

```
                                        read has completed */
  int   fr_selection;            /* Selection criteria of aggregates desired */
  unsigned int   fr_entries;     /* Number of entries returned
                                    by zFS (for OffsetCusor) */
  int   fr_nonFatalRc;           /* Non-fatal error code */
  int   fr_nonFatalRsn;          /* Reason code if fr_nonFatalRc is non-0 */
  char  fr_resumeName[45];  /* Dataset name to resume with for NameCursor or */
                            /* the name for the single-aggregate query.*/
  char  fr_patternName[45]; /* The pattern name to be used. */
  char  fr_future2[2];
} FSINFO_REQUEST;

typedef struct FSINFO_NAME_t
{
  char  fn_eye[4];
#define FN_EYE "FINA"
  short fn_slength;           /* Structure length */
  short fn_sversion;
  char  fn_name[44];          /* aggregate name */
  unsigned int  fn_connected; /* number of conneceted systems if owner
                                 output is included; 0 otherwise*/
  char  fn_owner[8];          /* system name of the owner */
  unsigned int fn_length;     /* Total length of all information for this
                                 aggregate, so programs can quickly find the
                                 beginning of the next record
                                 in the output buffer. */
  char  fn_future[4];
  char  fn_sysnames[8];       /* Names of connected systems (32 at most).Actual
                                 number is defined fn_connected.*/
} FSINFO_NAME;

typedef struct FSINFO_OWNER_t {
  char           fo_eye[4];
#define          FSO_EYE "FIOW"
  short          fo_length;
  short          fo_sversion;
  unsigned int   fo_size;          /* Num of 8K blocks in the aggregate */
  unsigned int   fo_free;          /* Number of unused 8K blocks
                                      in the aggregate.*/
  unsigned long long int fo_frags; /* Num of free 1K fragments
                                      available in the aggregate.*/
  unsigned int   fo_logsize;       /* Num of 8K blocks allocated
                                      to the log file for
                                      transaction logging,
                                      including indirect blocks.*/
  unsigned int   fo_bitmapsize;    /* Number of 8K blocks allocated to the
                                      bitmap file including indirect blocks.*/
  unsigned int   fo_anodesize;     /* Number of 8K blocks allocated
                                      to the anode table.*/
  unsigned int   fo_objects;       /* Number of objects in the file system. */
  char           fo_version;       /* Aggregate version number */
  char           fo_threshold;     /* Space monitoring threshold */
  char           fo_increment;     /* Space monitoring increment*/
  char           reserved1;
  int            fo_flags;
#define FO_OWNER_MNTRW       0x1   /* Mounted in RW mode */
#define FO_OWNER_DISABLED    0x2   /* Disabled for access */
#define FO_OWNER_GROWFAIL    0x4   /* Grow failure since last reset */
#define FO_OWNER_LOW_ONSPC   0x8   /* Low on space (zfs definition)*/
#define FO_OWNER_DAMAGED     0x10  /* Aggregate is damaged by salvage
                                      verification & not repaired yet */
#define FO_OWNER_RWSHARE     0x20  /* Aggregate using zFS sysplex
                                      sharing (RWSHARE) */
#define FO_OWNER_GROWSET     0x40  /* Dynamic grow set at mount time */
#define FO_OWNER_GROWING     0x80  /* Aggregate is in the process
                                      of growing at the time of query */
#define FO_CONVERTOV5        0x100 /* CONVERTTOV5 parm is set on mount. */
```

```
#define FO_NOTMOUNT          0x200 /* Aggregate is not mounted */
#define FO_NO_OWNER          0x400 /* Aggregate is un-owned */
#define FO_OWNER_ALLOWGROW   0x800 /* Dynamic grow allowed , no
                                    grow failures or since a grow
                                    failure an admin grow was done. */
#define FO_OWNER_CHGOWNER    0x1000     /* The quiesce is done for a
                                         chgowner instead of a backup */
#define FO_CONVERTTOV5_DISABLED 0x2000  /* CONVERTTOV5 is disabled
                                         due to quiesce or failed convert */
#define FO_V4                0x4000     /* Aggregate with version 1.4 */
#define FO_V5                0x8000     /* Aggregate with version 1.5 */

  unsigned int fo_overflow; /* Num of overflow pages used for v5 directories */
  unsigned int fo_overflowhiwater; /* Hiwater mark of fo_overflow
                                    for life of file system.*/
  unsigned int fo_thrashing;       /* Current number of objects using
                                    the thrash-resolution protocol*/
  char         reserved2[4];
  unsigned long long int fo_thrash_resolution; /* Number of thrash resolutions
                                                performed since last
                                                statistics reset.*/
  unsigned long long int fo_revocations; /* Number of token revocations
                                          performed since last
                                          statistics reset*/
  unsigned long long int fo_revwait;     /* Average revocation wait time
                                          in microseconds.*/
  char                   fo_qsysname[8]; /* Name of system requesting quiesce,
                                          if the aggregate is quiesced,
                                          0 otherwise.*/
  char                   fo_jobname[8];  /* Name of job requesting quiesce,
                                          if the aggregate is quiesced,
                                          0 otherwise.*/
  unsigned long long int fo_createtime;  /* Creation time in
                                          seconds since epoch*/
  unsigned long long int fo_ownership;   /* Owership time in
                                          seconds since epoch*/
  unsigned long long int fo_reset;       /* Time statistic counters reset in
                                          seconds since last epoch*/
  unsigned long long int fo_quiesce;     /* Quiesce time in seconds since
                                          epoch, 0 if file system
                                          not quiesced.*/
  unsigned int           fo_devno;       /* Devno for the mount*/
  char                   fo_auditfid[10]; /* Audit fid for file system*/
  unsigned short         fo_qasid;       /* ASID which issued the quiesce */
  unsigned int           fo_growcount;   /* Number of grows since mount. */
  char                   reserved3[4];
  unsigned long long int fo_growtime;    /* Time of the last grow
                                          as known by owner */
} FSINFO_OWNER;

typedef struct FSINFO_LOCAL_t {
  char                   fl_eye[4];
#define                  FL_EYE  "FILO"
  short                  fl_length;
  short                  fl_sversion;     /* Structure version */
  unsigned long long int fl_vnodes;       /* Number of vnodes cached in memory
                                           on the local system */
  unsigned long long int fl_ussheld;      /* Number of USS held vnodes*/
  char                   fl_sysname[8];   /* System name these stats are for */
  unsigned long long int fl_open;         /* Number of open objects in
                                           the file system */
  unsigned long long int fl_tokens;       /* Number of tokens held from
                                           the token manager */
  unsigned int           fl_usercache;    /* Number of 4K pages held in the
                                           user cache for the file system */
  unsigned int           fl_metacache;    /* Number of 8k pages held in
                                           the metadata cache */
  unsigned long long int fl_appreads;     /* Number of application reads made
                                             since last reset */
  unsigned long long int fl_appreadresp;  /* Average read response
                                             time in microseconds*/
  unsigned long long int fl_appwrites;    /* Number of application writes
                                             made since last reset */
  unsigned long long int fl_appwriteresp; /* Average write response
                                             time in microseconds*/
  unsigned long long int fl_xcfreads;     /* Number of xcf read calls made
                                             to the owner since last reset */
  unsigned long long int fl_xcfreadresp;  /* Average xcf read call response
                                             time in microseconds*/
  unsigned long long int fl_xcfwrites;    /* Number of xcf write calls made to
                                             the server since last reset */
  unsigned long long int fl_xcfwriteresp; /* Average xcf write call response
                                             time in microseconds*/
  unsigned long long int fl_enospc;       /* Number of ENOSPC errors returned
                                             to apps since last reset */
  unsigned long long int fl_ioerrs;       /* Number of disk IO errors
                                             since last reset*/
  unsigned long long int fl_commerrs;     /* Number of XCF communication timeouts
                                             or failures since last reset*/
  unsigned long long int fl_cancels;      /* Number of cancelled operations
                                             since last reset by asynchronus
                                             abends, cancel, forces and EOMs */
  char                   fl_ddname[8];    /* DDNAME of allocation of dataset */
  struct timeval64       fl_mounttime;    /* Mount time, seconds since epoch */
  unsigned int           fl_numdasd;      /* Number of DASD volumes listed for
                                             aggregate in FSINFO_DASD array */
  unsigned int           fl_flags;        /* 1 indicates if this system has
                                             tasks waiting on a quiesced FS.*/
} FSINFO_LOCAL;

typedef struct FSINFO_DASD_t
{
  char  fd_eye[4];
#define FSD_EYE  "FIDA"
  short fd_length;
  short fd_sversion;
#define FSD_VER_INITIAL 1
  char  fd_volser[6];
  short fd_pavios;
  unsigned long long int fd_reads;
  unsigned long long int fd_readbytes;
  unsigned long long int fd_writes;
  unsigned long long int fd_writebytes;
  unsigned long long int fd_waits;
  unsigned long long int fd_waitTime;
  unsigned long long int fd_resptime;
} FSINFO_DASD;

void check_local_error(char *buffp, FSINFO_REQUEST *fs_req, int *lerr_stat);

int main(int argc, char **argv)
{
  char*             buffp          = NULL;
  syscall_parmlist* parmp          = NULL;
  FSINFO_REQUEST*   fs_req         = NULL;
  char              owner_sys[9];

  int               buff_fill_len  = 0;
  int               fs_ownerlen    = 0;
  int               fs_locallen    = 0;
  int               unowned        = 0;
  int               fr_nonFatalRc  = 0;
  int               fr_nonFatalRsn = 0;
  int               sperr          = 0;
  int              ioerr         = 0;
  int              xcferr        = 0;
  int              lerr_stat     = 0;
  int              bpxrv, bpxrc, bpxrs;
  int              i, j, k;
  unsigned long long int most_writes = 0;
  char             busiest_volume[7];
  int              locals        = 0;

  /* aggrname for fsinfo */
  char             aggrname[ZFS_MAX_AGGRNAME+1] = "PLEX.DCEIMGQY.FS";

  /* Output structure pointers */
  FSINFO_NAME*     fs_namep      = NULL;
  FSINFO_OWNER*    fs_ownerp     = NULL;
  FSINFO_LOCAL*    fs_localp     = NULL;
  FSINFO_DASD *    fs_dasdp      = NULL;
  char*            outputp       = NULL;

  /* Allocate buffer */
  buffp = (char*) malloc(BUFFER_SIZE);
  if( buffp == NULL )
  {
    printf("Malloc Error\n");
    return 0;
  }

  /* Set the parmdata */
  parmp = (syscall_parmlist*) &buffp[0];
  parmp->opcode   = AGOP_FSINFO_PARMDATA;
  parmp->parms[0] = buff_fill_len = sizeof(syscall_parmlist);
  parmp->parms[1] = 0;
  parmp->parms[2] = 0;
  parmp->parms[3] = 0;
  parmp->parms[4] = 0;
  parmp->parms[5] = 0;
  parmp->parms[6] = 0;

  fs_req = (FSINFO_REQUEST*) &buffp[buff_fill_len];
  memset( fs_req, 0x00, sizeof(FSINFO_REQUEST) );

  /* First obtain the statistics for all file systems. We will look      */
  /* through them to find the DASD volume with the most write operations. */
  memcpy( fs_req->fr_eye, FR_EYE, sizeof(fs_req->fr_eye) );
  fs_req->fr_length        = sizeof(FSINFO_REQUEST);
  fs_req->fr_sversion      = 1;
  fs_req->fr_version       = 1;
  fs_req->fr_reqtype       = FR_REQTYPE_NAMECURSOR;
  fs_req->fr_output        = FR_OUT_FULL_STAT;
  fs_req->fr_nameSelection = FR_NM_ALLAGGR;

  buff_fill_len += sizeof(FSINFO_REQUEST);

  /* Loop getting file system information from zFS until we have it all. */
  do
  {
      /* Call zFS. */
      printf("call zfs\n");
      BPX1PCT("ZFS     ",
              ZFSCALL_FSINFO,             /* Aggregate operation */
              BUFFER_SIZE,                /* Length of Argument */
              (char*) buffp,              /* Pointer to Argument */
              &bpxrv,                     /* Pointer to Return_value */
              &bpxrc,                     /* Pointer to Return_code */
              &bpxrs);                    /* Pointer to Reason_code */

      if( bpxrv )
```

```
{
  printf("Error getting fsinfo for aggregate %s\n", aggrname);
  printf("Return Value: %d Return Code: %d Reason Code: %x\n",
         bpxrv, bpxrc, bpxrs);
  goto done;
}
if( fs_req->fr_nonFatalRc )
{
  fr_nonFatalRc  = fs_req->fr_nonFatalRc;
  fr_nonFatalRsn = fs_req->fr_nonFatalRsn;
  goto print_non_fatals;
}

/* The first structure pointed by output buffer is FSINFO_NAME.*/
fs_namep = (FSINFO_NAME *) &buffp[buff_fill_len];
for (i=0; i<fs_req->fr_entries; i++)
{
  fs_ownerp = (FSINFO_OWNER *)((char *)fs_namep+fs_namep->fn_slength);
  locals = fs_namep->fn_connected;

  /* If file system has an owner, there will be one more */
  /* FSINFO_LOCAL structure returned than this count.    */
  if (memcmp(fs_namep->fn_owner, "*UNOWNED") != 0)
    locals++;

  /* Determine if there is an FSINFO_OWNER or not.         */
  /* If not, then the structure should be an FSINFO_LOCAL. */
  if (memcmp(fs_ownerp->fo_eye, FSO_EYE, 4) == 0)
  { /* FSINFO_OWNER returned */
    fs_localp = (FSINFO_LOCAL *)((char *)fs_ownerp+fs_ownerp->fo_length);
  }
  else if (memcmp(fs_ownerp->fo_eye, FL_EYE, 4) == 0)
  {
    /* No FSINFO_OWNER returned. It's FSINFO_LOCAL */
    fs_localp = (FSINFO_LOCAL *)fs_ownerp;
    fs_ownerp = NULL;
  }
  else
  {
    /* Should not get here!! */
    printf("Error exit: Incorrect structure sequence!!\n");
    goto done;
  }

  /* Loop through each FSINFO_LOCAL structure returned. */
  for (j=0; j<locals; j++)
  {
    fs_dasdp = (FSINFO_DASD *)((char *)fs_localp + fs_localp->fl_length);
    for (k=0; k<fs_localp->fl_numdasd; k++)
    {
      /* Determine if this DASD volume has more writes than the  */
      /* previously higher one.  Yes, remember DASD volume name. */
      if (fs_dasdp->fd_writes > most_writes)
      {
        strncpy(busiest_volume, fs_dasdp->fd_volser, 6);
        busiest_volume[6] = 0;
        most_writes = fs_dasdp->fd_writes;
      }
      /* Set up for next iteration. */
      fs_dasdp = (FSINFO_DASD *)((char *)fs_dasdp + fs_dasdp->fd_length);
    }
    /* After looping through all FSINFO_DASD structures, fs_dasdp */
    /* should be pointing at the next FSINFO_LOCAL structure.     */
    fs_localp = (FSINFO_LOCAL *)fs_dasdp;
  }

  /* Get ready for next loop iteration. */
```

```
      fs_namep = (FSINFO_NAME *)((char *)fs_namep+fs_namep->fn_length);
    }
}
while (!fs_req->fr_eol);

printf("DASD volume %s has the most writes (%llu)\n",
       busiest_volume, most_writes);

/* Now do a single aggregate query for a specific file system. */
memset( fs_req, 0x00, sizeof(FSINFO_REQUEST));
memcpy( fs_req->fr_eye, FR_EYE, sizeof(fs_req->fr_eye) );
fs_req->fr_length        = sizeof(FSINFO_REQUEST);
fs_req->fr_sversion      = 1;
fs_req->fr_version       = 1;
fs_req->fr_output        = FR_OUT_FULL_STAT;
fs_req->fr_reqtype       = FR_REQTYPE_SINGLEQUERY;
memcpy( fs_req->fr_resumeName, aggrname, ZFS_MAX_AGGRNAME+1 );

BPX1PCT("ZFS     ",
        ZFSCALL_FSINFO,             /* Aggregate operation */
        BUFFER_SIZE,                /* Length of Argument */
        (char*) buffp,              /* Pointer to Argument */
        &bpxrv,                     /* Pointer to Return_value */
        &bpxrc,                     /* Pointer to Return_code */
        &bpxrs);                    /* Pointer to Reason_code */

if( bpxrv )
{
  printf("Error getting fsinfo for aggregate %s\n", aggrname);
  printf("Return Value: %d Return Code: %d Reason Code: %x\n",
         bpxrv, bpxrc, bpxrs);
  goto done;
}
if( fs_req->fr_nonFatalRc )
{
  fr_nonFatalRc  = fs_req->fr_nonFatalRc;
  fr_nonFatalRsn = fs_req->fr_nonFatalRsn;
  goto print_non_fatals;
}

buff_fill_len = sizeof(syscall_parmlist) + sizeof(FSINFO_REQUEST);
outputp = buffp + buff_fill_len;
check_local_error(outputp, fs_req, &lerr_stat);

/* The first structure pointed by output buffer would be FSINFO_NAME. */
fs_namep = (FSINFO_NAME *) &buffp[buff_fill_len];
fs_ownerp = (FSINFO_OWNER *) ((char*) fs_namep + fs_namep->fn_slength);
memcpy(owner_sys, fs_namep->fn_owner, 8);
owner_sys[8] = '\0';

if (memcmp(&owner_sys[0], "*UNOWNED", 8) == 0)
{
   unowned = 1;
   if (memcmp(fs_ownerp->fo_eye, FSO_EYE, 4) == 0)
   {  /* FSINFO_OWNER returned */
      fs_localp = (FSINFO_LOCAL *)((char *)fs_ownerp + fs_ownerp->fo_length);
   }
   else if (memcmp(fs_ownerp->fo_eye, FL_EYE, 4) == 0)
   {
      /* No FSINFO_OWNER returned. It's FSINFO_LOCAL */
      fs_localp = (FSINFO_LOCAL *)fs_ownerp;
      fs_ownerp = NULL;
   }
}
else if (fs_ownerp->fo_flags & FO_NO_OWNER)
{
   unowned = 1;
```

```
      fs_localp = (FSINFO_LOCAL *)((char *)fs_ownerp + fs_ownerp->fo_length);
   }
   else
      fs_localp = (FSINFO_LOCAL *)((char *)fs_ownerp + fs_ownerp->fo_length);

   if ((lerr_stat & FSINFO_SPC_ERR) == FSINFO_SPC_ERR)
   {
     fs_localp->fl_enospc = 1;
     sperr = 1;
   }
   if ((lerr_stat & FSINFO_IO_ERR) == FSINFO_IO_ERR)
   {
     fs_localp->fl_ioerrs = 1;
     ioerr = 1;
   }
   if ((lerr_stat & FSINFO_XCF_ERR) == FSINFO_XCF_ERR)
   {
     fs_localp->fl_commerrs = 1;
     xcferr = 1;
   }

   if( unowned && !fs_ownerp )
   {
     if (!xcferr && !ioerr && !sperr)
       printf("%-44.44s %-8.8s n/a \n\n",
              aggrname, "*UNOWNED");
     else
     {
       printf("%-44.44s %-8.8s %s%s%s \n\n",
              aggrname, "*UNOWNED",
              (sperr)? "SE" :"",
              (ioerr)?((sperr)?",IE":"IE"):"",
              (xcferr)?((sperr || ioerr)?",CE":"CE"):"");
       /* Define the flags in a legend */
       printf("Legend: %s%s%s\n\n",
              (sperr)? "SE = Space errors reported":"",
              (ioerr)?
                ((sperr)? ",IE = IO errors reported":
                          "IE = IO errors reported") : "",
              (xcferr)?
                ((sperr || ioerr)?
                          ",CE = Communication errors reported":
                          "CE = Communication errors reported") : "");
     }
   }
   else
   {
     /* Print the aggregate info with flags */
     printf("%-44.44s %-8.8s %s%s%s%s%s%s%s%s%s%s%s%s%s%s \n\n",
            aggrname, fs_namep->fn_owner,
            (fs_ownerp->fo_flags & FO_NOTMOUNT)         ? "NM" : "",
            /* Multiple Conditions */
            (!(fs_ownerp->fo_flags & FO_NOTMOUNT) &&
             (fs_ownerp->fo_flags & FO_OWNER_MNTRW))    ? "RW" :
            ((fs_ownerp->fo_flags & FO_NOTMOUNT)        ? ""   : "RO"),
            /* Multiple Conditions */
            (!(fs_ownerp->fo_flags & FO_NOTMOUNT) &&
             (fs_ownerp->fo_flags & FO_OWNER_RWSHARE))  ? ",RS" :
            ((fs_ownerp->fo_flags & FO_NOTMOUNT)        ? ""    : ",NS"),

            (fs_ownerp->fo_thrashing)                   ? ",TH" : "",
            (fs_ownerp->fo_qsysname[0] != '\0')         ? ",Q"  : "",
            (fs_ownerp->fo_flags & FO_OWNER_DISABLED)   ? ",DI" : "",
            (fs_ownerp->fo_flags & FO_OWNER_GROWING)    ? ",GR" : "",
            (fs_ownerp->fo_flags & FO_OWNER_GROWFAIL)   ? ",GF" : "",
            /* Multiple Conditions */
            (!(fs_ownerp->fo_flags & FO_NOTMOUNT) &&
             !(fs_ownerp->fo_flags & FO_OWNER_ALLOWGROW)) ? ",GD" : "",

            (fs_ownerp->fo_flags & FO_OWNER_DAMAGED)    ? ",DA" : "",
            (fs_ownerp->fo_flags & FO_OWNER_LOW_ONSPC)  ? ",L"  : "",
            (sperr)                                     ? ",SE" : "",
            (fs_ownerp->fo_flags & FO_OWNER_DISABLED)   ? ",DI" : "",
            (ioerr)                                     ? ",IE" : "",
            (xcferr)                                    ? ",CE" : "");


     /* Define the flags in a legend */
     printf("Legend: %s%s%s%s%s%s%s%s%s%s%s%s%s%s%s \n\n",
            (fs_ownerp->fo_flags & FO_NOTMOUNT)    ? "NM = Not mounted" : "",
            /* Multiple Conditions */
            (!(fs_ownerp->fo_flags & FO_NOTMOUNT) &&
             (fs_ownerp->fo_flags & FO_OWNER_MNTRW)) ? "RW = Read-write"  :
            ((fs_ownerp->fo_flags & FO_NOTMOUNT)     ? "" : "RO = Read-only"),
            /* Multiple Conditions */
            (!(fs_ownerp->fo_flags & FO_NOTMOUNT) &&
             (fs_ownerp->fo_flags & FO_OWNER_RWSHARE)) ?
             ",RS = Mounted RWSHARE" : ((fs_ownerp->fo_flags & FO_NOTMOUNT) ?
             "" : ",NS = Mounted NORWSHARE"),
            (fs_ownerp->fo_thrashing)              ? ",TH = Thrashing"  : "",
            (fs_ownerp->fo_qsysname[0] != '\0')     ? ",Q = Queisced"    : "",
            (fs_ownerp->fo_flags & FO_OWNER_DISABLED) ?
             ",DI = Disabled"   : "",
            (fs_ownerp->fo_flags & FO_OWNER_GROWING)  ?
             ",GR = Growing"    : "",
            (fs_ownerp->fo_flags & FO_OWNER_GROWFAIL) ?
             ",GF = Grow Failed": "",
            /* Multiple Conditions */
            (!(fs_ownerp->fo_flags & FO_NOTMOUNT) &&
             !(fs_ownerp->fo_flags & FO_OWNER_ALLOWGROW)) ?
               ",GD = AGGRGROW disabled" : "",
            (fs_ownerp->fo_flags & FO_OWNER_DAMAGED)  ?
             ",DA = Damaged"    : "",
            (fs_ownerp->fo_flags & FO_OWNER_LOW_ONSPC)  ?
             ",L = Low on space": "",
            (sperr)   ? ",SE = Space errors reported":"",
            (fs_ownerp->fo_flags & FO_OWNER_DISABLED)   ?
             ",DI = Disabled"   : "",
            (ioerr)   ? ",IE = IO errors reported"  : "",
            (xcferr)  ? ",CE = Communication errors reported":"");
   }
   goto done;

 print_non_fatals:
   if( fr_nonFatalRc )
   {
     printf("Non-Fatal errors:\n");
     printf("Return Code: %d Reason Code: %x\n\n",
            fr_nonFatalRc, fr_nonFatalRsn);
   }
 done:
   free(buffp);
   return 0;
 }

 void check_local_error(char *buffptr, FSINFO_REQUEST *fs_req, int *lerr_stat)
 {
   FSINFO_NAME * fs_namep;
   FSINFO_OWNER * fs_ownerp = NULL;
   FSINFO_LOCAL * fs_local;
   FSINFO_DASD  * dasdp;
   int dasd_space;
   int i, j;
   int total_sys = 0;
```

```
int unowned = 0;

if ((*lerr_stat) == (FSINFO_XCF_ERR | FSINFO_IO_ERR | FSINFO_SPC_ERR))
{
  printf("FSINFO_CheckLocalErr: all 3 bits are set in *lerr_stat=%X\n",
        *lerr_stat);
  return ;
}

/* The first structure pointed by output buffer would be FSINFO_NAME. */
fs_namep = (FSINFO_NAME *)((char *)buffptr);
fs_ownerp = (FSINFO_OWNER *)((char *)fs_namep + fs_namep->fn_slength);

/* if UNOWNED, make sure we are processing the right stats. */
if (memcmp(&fs_namep->fn_owner, "*UNOWNED", 8) == 0)
{
  unowned = 1;
  if (memcmp(fs_ownerp->fo_eye, FSO_EYE, 4) == 0)
  { /* FSINFO_OWNER block */
    fs_local = (FSINFO_LOCAL *)((char *)fs_ownerp + fs_ownerp->fo_length);
  }
  else if (memcmp(fs_ownerp->fo_eye, FL_EYE, 4) == 0)
  { /* FSINFO_LOCAL block */
    fs_local = (FSINFO_LOCAL *)((char *)fs_ownerp + fs_ownerp->fo_length);
    fs_ownerp = NULL;
  }
  else
  {  /* We should not get here!! */
    return;
  }
}
else
  fs_local = (FSINFO_LOCAL *)((char *)fs_ownerp + fs_ownerp->fo_length);

/* If FSINFO_OWNER is not returned, we have 1 less FSINFO_LOCAL to process */
if (unowned && (fs_ownerp == NULL))
  total_sys = fs_namep->fn_connected;
else
  total_sys = fs_namep->fn_connected+1;

for (i=0; i < total_sys; i++)
{
  if (fs_local->fl_commerrs)
    (*lerr_stat) |= FSINFO_XCF_ERR;

  if (fs_local->fl_enospc)
    (*lerr_stat) |= FSINFO_SPC_ERR;

  if (fs_local->fl_ioerrs)
    (*lerr_stat) |= FSINFO_IO_ERR;

  if ((*lerr_stat) == (FSINFO_XCF_ERR | FSINFO_IO_ERR | FSINFO_SPC_ERR))
    return ;

  /* Find the next FSINFO_LOCAL structure, which is after any FSINFO_DASD */
  /* structures that might be present.                                    */
  if (fs_local->fl_numdasd > 0)
  {
      dasdp = (FSINFO_DASD *)((char *)fs_local + fs_local->fl_length);
      dasd_space = fs_local->fl_numdasd * dasdp->fd_length;
  }
  else
      dasd_space = 0;
  fs_local = (FSINFO_LOCAL *)((char *)fs_local + fs_local->fl_length +
                              dasd_space);
        }

        return;
      }
```

List File Information

Purpose

Lists detailed file or directory information. This API is an **w_pioctl** (BPX1PIO) call specifying a path name rather than a **pfsctl** (BPX1PCT) call specifying a file system name.

Format

```
BPX1PIO parameter list
   Pathname_length     int
   Pathname            char[1025]
   Command             int                 0x0000A901
   Argument_length     int                 sizeof(FOBJ_INFO)
   Argument            ptr to FOBJ_INFO
   Return_value        ptr to int          0
   Return_code         ptr to int          0
   Reason_code         ptr to int          0

FOBJ_TIME
   fo_seconds          hyper               Second since last epoch
   fo_mircoseconds     int                 Micro seconds since last epoch
   fo_unused           int                 Reserved

FOBJ_ACLINFO
   fo_index            int                 Location of ACL
   fo_length           int                 Length of ACL

FOBJ_AUDIT
   fo_read             char                Read information
   fo_write            char                Write information
   fo_exec             char                Exec information
   fo_res1             char                1 - No auditing
                                           2 - Success auditing
                                           3 - Failure auditing

FOBJ_SYSINFO
   fo_vnode            hyper               Address of zFS vnode
   fo_vntok            hyper               Address of z/OS UNIX vnode
   fo_openwaiters      int                 Number of tasks waiting to open a
                                           file blocked by deny-mode opens
   fo_internalopens    int                 Number of internal opens
   fo_readopens        int                 Number of opens for read
   fo_writeopens       int                 Number of opens for write
   fo_denyreads        short               Number of deny-read opens
   fo_denywrites       short               Number of deny-write opens
   fo_advdebyreads     short               Number of advisory deny-read opens
   fo_advdenywrites    short               Number of advisory deny-write opens
   fo_sysflags         char                Miscellaneous information:
                                           0x01 - file being read sequentially
                                           0x02 - file written sequentially
                                           0x04 - security information cached
                                           0x08 - file location information
                                                  cached
                                           0x10 - symlink information cached
                                           0x20 - metadata updates sent to
                                                  server, can not directly
                                                  read without a server sync
                                           0x40 - tokens are being revoked
                                           0x80 - file is undergoing thrashing
   fo_sysflags2       char               More miscellaneous information
                                         0x01 - file system owned locally
   fo_unused          char[2]            Reserved
   fo_unscheduled     int                Number of 4K pages in user file
                                         cache that need to be written
   fo_pending         int                Number of 4K pages being written
   fo_segments        int                Number of 64K segments in user cache
   fo_dirtysegment    int                Number of segments with pages that
                                         need to be written
   fo_metaissued      int                Number of I/Os in progress that will
                                         require a metadata update
   fo_metapending     int                Number of queued metadata updates
   fo_rights          int                Token rights held by object
   fo_xmits           short              Number of XCF messages client has
                                         sent server for this object
   fo_fwd             short              Number of in-progress operations
                                         for object using thrashing protocol
   fo_metabuffers     int                Number of buffers in metadata cache
                                         for this object, only client systems
   fo_dirtybuffers    int                Number of metadata buffers updated
                                         for object that are on server and
                                         need writing
   fo_owner           char[9]            Name of owning system
   fo_localsys        char[9]            Name of local system
   fo_pad             char[2]            Reserved
   fo_sysres          int[9]             Reserved

FOBJ_INFO
   fo_eye             char[4]            "FOIN"
   fo_len             short              Size of(FOBJ_INFO)
   fo_ver             char               1
   fo_inflags         char               1- Only in-memory system information
                                            is being requested.
   fo_inode           int                Object inode
   fo_unique          int                Object uniquifier
   fo_length          hyper              POSIX length of object (in bytes)
   fo_mtime           FOBJ_TIME          Last modification time
   fo_atime           FOBJ_TIME          Last access time
   fo_ctime           FOBJ_TIME          Last change time
   fo_reftime         FOBJ_TIME          Last reference time
   fo_create          FOBJ_TIME          Create time
   fo_allocation      char               How object stored on disk:
                                         1 - Object is stored inline
                                         2 - Object is stored fragmented
                                         3 - Object is stored blocked
   fo_owner_perms     char               Permissions for owner of file:
                                         0x01 - Execute permission
                                         0x02 - Write permission
                                         0x04 - Read permission
   fo_group_perms     char               Permissions for the group:access
                                         to the file:
                                         0x01 - Execute permission
                                         0x02 - Write permission
                                         0x04 - Read permission
   fo_other_perms     char               Permissions of other users of file:
                                         0x01 - Execute permission
                                         0x02 - Write permission
                                         0x04 - Read permission
```

```
  fo_allocated       unsigned int       Number of allocated bytes
  fo_locinfo         union              Location of object's data
      fo_direct      unsigned int[8]    Location of first 8 logical blocks
      fo_indirect    unsigned int[4]    Location of indirect tree roots
      -- or --
      fo_block       unsigned int       Block with object's data
      fo_start       unsigned short     Starting fragment in block
      fo_len         unsigned short     Number of fragments
  fo_uid             int                UID of owner
  fo_gid             int                GID of owner
  fo_access          FOBJ_ACLINFO       Access acl
  fo_dmodel          FOBJ_ACLINFO       Directory model acl
  fo_fmodel          FOBJ_ACLINFO       File model acl
  fo_user            FOBJ_AUDIT         User audit information
  fo_auditor         FOBJ_AUDIT         Auditor audit information
  fo_permbits        char               Sticky bit and other bits:
                                        0x01 - setgid
                                        0x02 - setuid
                                        0x04 - Sticky bit on
 <some bits>         int                Miscellaneous bits in an integer
  fo_txtflag         bit 0              Context are pure text
  fo_deferflag       bit 1              Defer tag set until first write
  fo_filefmt         bits 2-7           File format attribute:
                                        0=NA
                                        1=BIN
                                        2=NL
                                        3=CR
                                        4=LF
                                        5=CRLF
                                        6=LFCR
                                        7=CRNL
                                        8=REC
                     bits 8-31          Reserved
  fo_ccsid           unsigned short     Hex CCSID
  fo_seclabel        char[8]            Seclabel of object
  fo_entrycount      unsigned int       If object a directory, the number
                                        of names it contains.
  fo_linkcount       unsigned int       POSIX linkcount for object
  fo_dataversion     unsigned int       Data version for directory updates
  fo_genvalue        unsigned int       USS attribute flags of object
  fo_cver            char[8]            Creation verifier
  fo_majorminor      char[8]            If object a character special file,
                                        major/minor number.
  fo_type            char               Object type:
                                        0x01 - directory
                                        0x02 - regular file
                                        0x03 - symlink
                                        0x04 - FIFO
                                        0x05 - character special file
  fo_flags           char               Additional object flags:
                                        0x01 - object is a v5 directory
                                        0x02 - v5 directory tree structure
                                               is broken
                                        0x04 - automatic conversion to v5
                                               failed
                                        0x08 - contents are logged
  fo_offset          short              Offset of anode
  fo_anodeblock      unsigned int       Physical block that contains anode
  fo_status_level    char               Directory status byte
                                        0x80 - directory is v5
                                        0x1F - max depth of v5 tree
  fo_res             char[3]            Reserved
  fo_res3            int[14]            Reserved
  fo_info            FOBJ_SYSINFO       System based transient information
```

```
Return_value 0 if request is successful, -1 if it is not successful

Return_code
EBUSY            Aggregate containing file system is quiesced
EINTR            ZFS is shutting down
EINVAL           Invalid parameter list
EMVSERR          Internal error using an osi service
ENOENT           No such file or directory exists

Reason_code
0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
```

Usage notes

- The aggregate must be mounted or attached.
- If you set fo_inflags to 1, only local data is retrieved. If you set fo_inflags to 0, both global and local data are retrieved.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer requires lookup authority (x) to the directory and READ authority (r) to the file.

Related services

List Aggregate Status (Version 2)

Restrictions

None.

Examples

```
#pragma linkage(BPX1GCW, OS)
#pragma linkage(BPX1PIO, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1GCW(int, char *, int *, int *, int *);
extern void BPX1PIO(int, char *, int, int, void *, int *, int *, int *);

#include <stdio.h>
#include <time.h>

#define ZFSIOCTL_FILEINFO 0x0000A901    /* zFS ioctl command to        */
                                        /* return detailed fileinfo    */
                                        /* for a zFS file or directory */

#define hiszero(a) ((a).low == 0 && (a).high == 0)
#define hcmp(a,b)  ((a).high<(b).high? -1 : ((a).high > (b).high? 1 :    \
                   ((a).low <(b).low? -1 : ((a).low > (b).low? 1 : 0))))

#define u_int unsigned int
#define uint16_t unsigned short

typedef struct hyper {                  /* This is a 64 bit integer to zFS */
    unsigned int  high;
    unsigned int  low;
} hyper;

/****************************************************************************/
/* The FOBJ_INFO structure is used to contain the output of the fileinfo */
/* ioctl query to provide detailed information for a singular object in a */
/* zFS file system. */
/*************************************************************************/
typedef struct FOBJ_ACLINFO_t {
    int           fo_index;             /* Index into the anode table of */
                                        /* the location of the ACL       */
    int           fo_length;            /* Length of the ACL */
} FOBJ_ACLINFO;

typedef struct FOBJ_AUDIT_t {
    char          fo_read;              /* read auditing information */
    char          fo_write;             /* write auditing information */
    char          fo_exec;              /* exec auditing information */
    char          fo_res1;
#define FO_NONE 0                       /* no auditing */
#define FO_SUCC 1                       /* success auditing */
#define FO_FAIL 2                       /* fail auditing */
} FOBJ_AUDIT;

typedef struct FOBJ_TIME_t {
    hyper         fo_seconds;           /* number of seconds since epoch */
    int           fo_microseconds;      /* number of microseconds since epoch*/
    int           fo_tres1;             /* unused */
} FOBJ_TIME;

typedef struct FOBJ_SYSINFO_t {         /* HEX displacement into FOBJ_INFO */
    hyper         fo_vnode;         /* 138 - Address of vnode in zFS
                                       kernel memory */
    hyper         fo_vntok;         /* 140 - Address of USS vnode in
                                       z/OS Unix address space */
    int           fo_openwaiters;   /* 148 - Number of tasks waiting to open
                                       file because blocked
                                       by current deny-mode opens */
    int           fo_internalopens; /* 14C - Number of internal
                                       opens on the file */
    int           fo_readopens;     /* 150 - Number of opens for
                                       read on the file */
    int           fo_writeopens;    /* 154 - Number of write opens */
    short         fo_denyreads;     /* 158 - Number of deny-read opens */
    short         fo_denywrites;    /* 15A - Number of deny-write opens */
    short         fo_advdenyreads;  /* 15C - Number of adv. deny read opens */
    short         fo_advdenywrites; /* 15E - Number of adv. deny write opens */
    char          fo_sysflags;      /* 160 - Misc. information */
#define FO_SEQREAD 1                /* Object is a file that zFS determined
                                       is being read sequentially */
#define FO_SEQWRITE 2               /* Object is a file that zFS is
                                       being written sequentially */
#define FO_FSPVALID 4               /* System has security information
                                       cached for anode */
#define FO_ANODEVALID 8             /* System has posix attribute and
                                       disk location information cached */
#define FO_SYMLINKVALID 16          /* System has the symbolic link contents
                                       cached for the object */
#define FO_METAUPDATES 32           /* Client has sent metadata updates to the
                                       server, and cannot directly read without
                                       a server sync */
#define FO_REVOKE 64                /* Revoke in progress */
#define FO_THRASH 128               /* Object is considered sysplex-thrashing
                                       and thrash resolution is in
                                       effect for file */
    char          fo_sysflags2;     /* 161 - Misc. information 2 */
#define FO_OWNER 1                  /* This system is the owner of
                                       the file system */
    char          fo_unused[2];     /* 162 - reserved */
    int           fo_unscheduled;   /* 164 - Number of dirty 4K pages in the
                                       user file cache that have not yet been
                                       written to disk */
    int          fo_pending;       /* 168 - Number of pending 4K pages
                                      in transit to disk */
    int          fo_segments;      /* 16C - Number of 64K segment structures
                                      in the user file cache for the file */
    int          fo_dirtysegments; /* 170 - Number of 64K segment structures
                                      that have dirty pages in the
                                      user file cache */
    int          fo_metaissued;    /* 174 - Number of in-progress IOs to disk
                                      that will require a metadata
                                      update to reflect new data in the file*/
    int          fo_metapending;   /* 178 - Number of queued metadata updates
                                      for file, for IOs completed to new data
                                      for the file */
    int          fo_rights;        /* 17C - Token rights held for object */
    short        fo_xmits;         /* 180 - Number of in-progress
                                      transmissions from client to
                                      server for this file */
    short        fo_fwd;           /* 182 - Number of in-progress forwarded
                                      operations due to thrashing object */
    int          fo_metabuffers;   /* 184 - Number of buffers for file in the
                                      metadata cache - client only */
    int          fo_dirtybuffers;  /* 188 - Number of dirty metadata buffers
                                      in the metadata cache for
                                      object - server only */
    char         fo_owner[9];      /* 18C - the name of the owner */
    char         fo_localsys[9];   /* 195 - the name of the local system */
    char         fo_pad[2];        /* 19E - pad */
#define FO_SYSRES_NUM 9
    int          fo_sysres[FO_SYSRES_NUM]; /* 1A0 - Reserved for future use */
} FOBJ_SYSINFO;

typedef struct fobj_info_t {       /* HEX displacement into FOBJ_INFO */
    char         fo_eye[4];        /* 000 - Eye catcher */
#define FO_EYE "FOIN"
    short        fo_len;           /* 004 - Length of this structure */
    char         fo_ver;           /* 006 - Version */
#define FO_VER_INITIAL 1          /* Initial version */
    char         fo_inflags;      /* 007 - Input flag bits indicating
                                     requested function */
#define FO_SYSINFO_ONLY 1         /* Only the in-memory system information
                                     is being requested */
    int          fo_inode;        /* 008 - Inode of the object */
    int          fo_unique;       /* 00C - Uniquifier of the object */
    hyper        fo_length;       /* 010 - Posix length of object in bytes */
    FOBJ_TIME    fo_mtime;        /* 018 - Modification time */
    FOBJ_TIME    fo_atime;        /* 028 - access time */
    FOBJ_TIME    fo_ctime;        /* 038 - change time */
    FOBJ_TIME    fo_reftime;      /* 048 - referenct time */
    FOBJ_TIME    fo_create;       /* 058 - creation time of object */
    char         fo_allocation;   /* 068 - How the object is stored on disk */
#define FO_INLINE 1               /* Object is stored inline */
#define FO_FRAGMENTED 2           /* Object is stored fragmented */
#define FO_BLOCKED 3              /* Object is stored in the blocked
                                     method, or is empty */
    char         fo_owner_perms;  /* 069 - Permissions for the owner
                                     of this file */
#define FO_READ 4                 /* has read permission */
#define FO_WRITE 2                /* has write permission */
#define FO_EXEC 1                 /* has execute permission */
    char         fo_group_perms;  /* 06A -Permissions for the group
                                     associated with this file */
    char         fo_other_perms;  /* 06B - Permissions for other.. */
    unsigned int fo_allocated;    /* 06C - Number of allocated bytes to
                                     object, including internal control
                                     structures, in kilobyte units */
    union
    {
        struct {
            unsigned int fo_direct[8];   /* 070 - Physical location of first 8
                                            logical blocks of object */
            unsigned int fo_indirect[4]; /* 090 - Physical location of indirect
                                            tree roots, trees 0 - 3 */
#define     FO_UNALLOCATED 0xFFFFFFFF    /* This value means block is not
                                            allocated in fo_direct or
                                            fo_indirect slot */
        } fo_blockinfo;

        struct {
            unsigned int   fo_block;    /* 070 - Block that contains the
                                           object data */
            unsigned short fo_start;    /* 074 - Start fragment in the block */
            unsigned short fo_len;      /* 076 - Number of fragments
                                           in the block */
        } fo_fraginfo;
    } fo_locinfo;                       /* Location of objects data */

    int fo_uid;                         /* 0A0 - UID of the owner of object */
    int fo_gid;                         /* 0A4 - group id of owner of object */
    FOBJ_ACLINFO fo_access;             /* 0A8 - ACL information for access
                                           acl of object */
    FOBJ_ACLINFO fo_dmodel;             /* 0B0 - ACL information for directory
                                           model acl */
    FOBJ_ACLINFO fo_fmodel;             /* 0B8 - ACL information for file
                                           model acl */
    FOBJ_AUDIT fo_user;                 /* 0C0 - User auditing information */
    FOBJ_AUDIT fo_auditor;              /* 0C4 - Auditor auditing information*/
    char fo_permbits;                   /* 0C8 - Sticky and other bits */
#define FO_ISVTX 4                      /* sticky bit on */
#define FO_ISUID 2                      /* setuid */
#define FO_ISGID 1                      /* setgid */
    int fo_txtflag : 1;                 /* 0C9 - contents are pure
                                           text indicator */
    int fo_defertag : 1;                /* 0C9 - Defer tag set until
                                           first write */
    int fo_filefmt : 6;                 /* 0C9 - File format attribute */
                                        /* 0=NA 1=BIN 2=NL 3=CR 4= LF */
                                        /* 5=CRLF 6=LFCR 7=CRNL 8=REC */
    short        fo_ccsid;              /* 0CA - hex ccsid */
    char         fo_seclabel[8];        /* 0CC - seclabel of the object */
    unsigned int fo_entrycount;         /* 0D4 - Number of names in the
                                           directory, if this is a directory */
    unsigned int fo_linkcount;          /* 0D8 - Posix linkcount for object */
    unsigned int fo_dataversion;        /* 0DC - Data version for
                                           directory updates */
    unsigned int fo_genvalue;           /* 0E0 - USS attribute flags
                                           of object */
    char         fo_cver[8];            /* 0E4 - Creation verifier */
    char         fo_majorminor[8];      /* 0EC - Major/minor number if object
                                           is a char special file */
    char         fo_type;               /* 0F4 - Object type */
#define FO_DIR 1                        /* object is directory */
#define FO_FILE 2                       /* object is a regular file */
#define FO_LINK 3                       /* object is a symlink */
#define FO_FIFO 4                       /* object is a fifo */
#define FO_CHARSPEC 5                   /* object is a char special file */
    char         fo_flags;              /* 0F5 - Additional flag bits of
                                           object */
#define FO_VER5 1                       /* Object is a directory stored in
                                           new-fast format */
#define FO_BROKEN 2                     /* The tree structure of this new-fast
                                           format dir is broken */
#define FO_CONVERT_FAIL 4               /* Automatic conversion of the
                                           directory failed */
    short        fo_offset;             /* 0F6 - Offset into the physical block
```

```
                                                   that contains the anode for object*/
      unsigned int fo_anodeblock;         /* 0F8 - Physical block in aggregate
                                             that contains the anode */
      char         fo_statuslevel;        /* 0FC - directory status byte */
      char         fo_res[3];             /* 0FD - reserved */
      int          fo_res3[14];           /* 100 - For future use */
      FOBJ_SYSINFO fo_info;               /* 138 - System based transient
                                             information */
  } FOBJ_INFO;                            /* 1C4 total length */

  int main(int argc, char **argv)
  {
    int       bpxrv;
    int       bpxrc;
    int       bpxrs;
    char      parm_pathname[1024];
    char      pathname[1024];
    char      *pathp          = NULL;
    FOBJ_INFO fobj;
    FOBJ_INFO *fo             = &fobj;
    void      *arg            = (void *)fo;
    int       arglen          = sizeof(fobj);
    char      buffer1[80];
    char      buffer2[80];
    hyper     bogusSignedTime;
    char      *p;
    char      *timep;
    char      time1_string[30];
    char      time2_string[30];
    char      seclabel[9];
    char      temp;

    if (argc < 2)
    {
      printf("Please specify a file or directory path name as a parameter\n");
      exit(1);
    }

    strncpy(parm_pathname, argv[1], sizeof(pathname));

    if (parm_pathname[0] == '/')  /* if absolute pathname         */
      pathp = parm_pathname;      /* put ptr to pathname in pathp */
    else
    {   /* if relative pathname */
      pathname[0] = 0;
      bpxrc = 0;
      bpxrv = 0;
      bpxrs = 0;

      /* get current working directory path */
      BPX1GCW(sizeof(pathname), pathname, &bpxrv, &bpxrc, &bpxrs);
      if (bpxrv == -1)
      {
        printf("BPX1GCW call failed rc %u rsn %8.8X\n", bpxrc, bpxrs);
        return bpxrc;
      }
      if ((strlen(pathname) + strlen(parm_pathname) + 1) > sizeof(pathname))
      {   /* if name longer than maximum pathname */
        printf("directory path name too long - input name len "
               "%d plus cwd len %d for buffer size %d\n",
               strlen(parm_pathname), strlen(pathname), sizeof(pathname));
        return 121; /* EINVAL */
      }

      /* take the current working directory and append slash */
      strcat(pathname, "/");
      /* then append the input relative path name */
      strcat(pathname, parm_pathname);
      /* put ptr to result in pathp */
      pathp = pathname;
    }

    bpxrc = 0;
    bpxrv = 0;
    bpxrs = 0;

    memset((char *)&fobj, 0x00, sizeof(fobj));
    memcpy(&fobj.fo_eye, FO_EYE, 4);
    fobj.fo_len = sizeof(fobj);
    fobj.fo_ver = FO_VER_INITIAL;
    BPX1PIO(strlen(pathp), pathp, ZFSIOCTL_FILEINFO,
            arglen, arg, &bpxrv, &bpxrc, &bpxrs);

    if (bpxrv < 0)
    {
      printf("Error getting fileinfo for pathname %s\n", pathp);
      printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
      return bpxrc;
    }
    else
    {   /* Return from fileinfo was successful */
      printf(" Object path: %s\n", pathp);
      printf(" Inode is %lu\n", fo->fo_inode);
      printf(" Length is %llu\n", fo->fo_length);

      /* Some common object information */
      printf(" Object type is %s\n",
             fo->fo_type == FO_DIR ? "DIR" :
             fo->fo_type == FO_FILE ? "FILE" :
             fo->fo_type == FO_LINK ? "LINK" :
             fo->fo_type == FO_CHARSPEC ? "CHARSPEC" : "??");

      /* Some directory object information */
      if (fo->fo_type == FO_DIR)
        printf(" Directory version %u\n",
               fo->fo_flags & FO_VER5 ? 5 : 4);
    }
    printf("\n");
    return 0;
  }
```

List File System Names (Version 1)

Purpose

Returns the names of the file systems contained in a specified aggregate on this system; the aggregate must be attached.

IBM recommends that you should use the List Detailed File System Information API instead of List Aggregate Status or List File System Status.

Format

```
syscall_parmlist
   opcode               int              138       AGOP_LISTFSNAMES_PARMDATA
   parms[0]             int              offset to AGGR_ID
   parms[1]             int              buffer length or 0
   parms[2]             int              offset to buffer or 0
   parms[3]             int              offset to size
   parms[4]             int              0
   parms[5]             int              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 int
     low                unsigned int
   fsid_aggrname        char[45]
   fsid_name            char[45]
   fsid_reserved        char[32]
   fsid_reserved2       char[2]
size                    int

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 notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

List Attached Aggregate Names
List Detailed File System Information
List File System Status

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_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 hyper {                     /* This is a 64 bit integer to zFS */
  unsigned int high;
  unsigned int low;
} hyper;

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 */
  int              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;
  int                total_fs;
  char               aggrname[45] = "PLEX.DCEIMGQX.FS";

  /* Ensure reserved fields are 0 */
  memset(&myparmstruct.aggr_id, 0, sizeof(AGGR_ID));
  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 = myp->size;         /* Get buffer size needed */
      mypsize = buflen +
                sizeof(syscall_parmlist) +
                sizeof(AGGR_ID) +
                sizeof(int);

      myp = (struct parmstruct *)malloc((int)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_PARMDATA;
      myp->myparms.parms[0] = sizeof(syscall_parmlist);
      myp->myparms.parms[1] = buflen;
      myp->myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
      myp->myparms.parms[3] = sizeof(syscall_parmlist) +
                              sizeof(AGGR_ID) +
                              buflen;
```

```
        myp->myparms.parms[4] = 0;
        myp->myparms.parms[5] = 0;
        myp->myparms.parms[6] = 0;

        BPX1PCT("ZFS     ",
                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\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

An aggregate operation that returns the names of the zFS file systems that are contained in a specified aggregate on this system and their corresponding z/OS UNIX file system names (if they are mounted). The specified aggregate must be attached.

IBM recommends using the List Detailed File System Information API instead of List Aggregate Status or List File System Status.

Format

```
syscall_parmlist
   opcode                int             144       AGOP_LISTFSNAMES_PARMDATA2
   parms[0]              int             offset to AGGR_ID
   parms[1]              int             buffer length or 0
   parms[2]              int             offset to buffer or 0
   parms[3]              int             offset to size
   parms[4]              int             0
   parms[5]              int             0
   parms[6]              int             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 int
     low                 unsigned int
   fsid_aggrname         char[45]
   fsid_name             char[45]
   fsid_mtname           char[45]
   fsid_reserved         char[49]
size                     int

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 notes

- The version 2 List File System Names returns an array of FS_ID2s.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

List Attached Aggregate Names
| List Detailed File System Information
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

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_LISTFSNAMES_PARMDATA2 144
#define E2BIG 145

typedef struct syscall_parmlist_t {
  int           opcode;        /* Operation code to perform */
  int           parms[7];      /* Specific to type of operation, */
                               /* provides access to the parms */
                               /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

#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
  char          aid_name[ZFS_MAX_AGGRNAME+1];   /* aggr name,null terminated */
  char          aid_reserved[33];               /* Reserved for the future */
} AGGR_ID;

typedef struct hyper {                          /* 64 bit integer to zFS */
  unsigned int  high;
  unsigned int  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
  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 {
  syscall_parmlist myparms;
  AGGR_ID          aggr_id;

  /* Real malloc'd structure will have an array of FS_ID2s here */
  int              size;
};

int main(int argc, char **argv)
{
  int               buffer_success = 0;
  int               bpxrv;
  int               bpxrc;
  int               bpxrs;
  int               t;
  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] = "PLEX.DCEIMGQX.FS";
  int               *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 */

  for(t = 0; t < 1000 && buffer_success == 0; t++)
  {
    if (bpxrv < 0)
    {
      if (bpxrc == E2BIG)
      {
        buflen = myp->size;               /* Get buffer size needed */
        mypsize = buflen +
                  sizeof(syscall_parmlist) +
                  sizeof(AGGR_ID) +
                  sizeof(myparmstruct.size);
        free(myp);

        myp = (struct parmstruct *)malloc((int)mypsize);
        memset(myp, 0, mypsize);
        memcpy(myp->aggr_id.aid_eye, AID_EYE, 4);
        myp->aggr_id.aid_len = sizeof(AGGR_ID);
        myp->aggr_id.aid_ver = AID_VER_INITIAL;
        strcpy(myp->aggr_id.aid_name, aggrname);

        myp->myparms.opcode = AGOP_LISTFSNAMES_PARMDATA2;
        myp->myparms.parms[0] = sizeof(syscall_parmlist);
        myp->myparms.parms[1] = buflen;
        myp->myparms.parms[2] = sizeof(syscall_parmlist) + sizeof(AGGR_ID);
        myp->myparms.parms[3] = sizeof(syscall_parmlist) +
                                sizeof(AGGR_ID) + buflen;
        myp->myparms.parms[4] = 0;
        myp->myparms.parms[5] = 0;
        myp->myparms.parms[6] = 0;

        BPX1PCT("ZFS     ",
                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 && bpxrc == E2BIG )
          printf("E2BIG: %d times total\n", t++);
        else if( bpxrv == 0 )
        {
          buffer_success = 1;
          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_mtname);
          }
          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");

      free(myp);
      return bpxrc;
    }
  }

  if( t == 1000 )
    printf("Number of failed buffer resizes exceeded.\n");

  free(myp);
  return 0;
}
```

List File System Status

Purpose

Lists status information of a file system. As input, use an FS_ID or an FS_ID2, which specifies the z/OS UNIX file system name (the mount name). For an FS_ID2, the file system must be mounted using that z/OS UNIX file system name. The aggregate that contains the file system must be attached and the aggregate cannot be quiesced.

| IBM recommends that you should use the List Detailed File System Information
| API instead of List Aggregate Status or List File System Status.

Format

```
syscall_parmlist
   opcode                    int            142    FSOP_GETSTAT_PARMDATA
   parms[0]                  int            Offset to FS_ID
   parms[1]                  int            Offset to FS_STATUS
   parms[2]                  int            0
   parms[3]                  int            0
   parms[4]                  int            0
   parms[5]                  int            0
   parms[6]                  int            0
FS_ID or FS_ID2
   fsid_eye                  char[4]        "FSID"
   fsid_len                  char           sizeof(FS_ID)
   fsid_ver                  char           1
   fsid_res1                 char           Reserved
   fsid_res2                 char           Reserved
   fsid_id
     high                    unsigned int   High portion of generated ID
     low                     unsigned int   Low portion of generated ID
   fsid_aggrname             char[45]       Aggregate name
   fsid_name                 char[45]       File system name
   fsid_reserved             char[32]       Reserved
   fsid_reserved2            char[2]        Reserved
FS_ID2 or FS_ID
   fsid_eye                  char[4]        "FSID"
   fsid_len                  char           sizeof(FS_ID2)
   fsid_ver                  char           2
   fsid_res1                 char           Reserved
   fsid_res2                 char           Reserved
   fsid_id
     high                    unsigned int   High portion of generated ID
     low                     unsigned int   Low portion of generated ID
   fsid_aggrname             char[45]       Aggregate name
   fsid_name                 char[45]       File system name
   fsid_mtname               char[45]       Name used when mounted
   fsid_reserved             char[49]       Reserved
FS_STATUS
   fs_eye                    char[4]        "FSST"
   fs_len                    short          sizeof(FS_STATUS)
   fs_ver                    char           1
   fs_res1                   char           Reserved
   fs_id
     high                    unsigned int   High portion of generated ID
     low                     unsigned int   Low portion of generated ID
   fsid_aggrname             char[45]       Aggregate name
   fsid_name                 char[45]       File system name
   fsid_mtname               char[45]       Name used when mounted
   fsid_reserved             char[49]       Reserved
```

```
fs_cloneTime               timeval         Time file system cloned
   fs_createTime              timeval         Time file system created
   fs_updateTime              timeval         Time of last update
   fs_accessTime              timeval         Time of last access
   fs_allocLimit              unsigned int    Number of blocks available
   fs_allocUsage              unsigned int    Number of blocks in use
   fs_visQuotaLimit           unsigned int    Quota for file system
   fs_visQuotaUsage           unsigned int    Blocks used in file system
   fs_accError                unsigned int    Error for invalid operation
   fs_accStatus               int             Operations being performed
   fs_states                  int             File system state
   fs_nodeMax                 int             Maximum inode number
   fs_minQuota                int             Minimum inode number
   fs_type                    int             Type of file system
   fs_threshold               char            FSFULL threshold monitoring
   fs_increment               char            FSFULL monitoring increment
   fs_mountstate              char            Mount status
                                              0 - Not mounted
                                              1 - Mounted R/W
                                              2 - Mounted readonly
   fs_msglen                  char            Length of status message
   fs_msg                     char[128]       Status message
   fs_aggrname                char[45]        Aggregate name
   fs_reserved1               char[3]         Reserved
   fs_reserved2               unsigned int[3] Reserved
   fs_InodeTbl                unsigned int    Size of Inode table
   fs_requests
     high                     unsigned int    High portion of number of file
                                              system requests by applications
     low                      unsigned int    Low portion of number of file
                                              system requests by applications
   fs_reserved3               unsigned int    Reserved
   fs_reserved4               unsigned int    Reserved
   fs_reserved5               unsigned int    Reserved
   fs_diskFormatMajorVersion  unsigned int    Major version of disk format
   fs_diskFormatMinorVersion  unsigned int    Minor version of disk format
   fs_reserved                char[80]        Reserved

- OR
-FS_STATUS2
   fs_eye                     char[4]         "FSST"
   fs_len                     short           sizeof(FS_STATUS)
   fs_ver                     char            2
   fs_res1                    char            Reserved
   fs_id
     high unsigned            int             High file system identifier
     low unsigned             int             Low file system identifier
   fs_cloneTime               timeval         Time file system cloned
   fs_createTime              timeval         Time file system created
   fs_updateTime              timeval         Time of last update
   fs_accessTime              timeval         Time of last access
   fs_allocLimit              unsigned int    Number of blocks available
   fs_allocUsage              unsigned int    Number of blocks in use
   fs_visQuotaLimit           unsigned int     Quota for file system
   fs_visQuotaUsage           unsigned int     Blocks used in file system
   fs_accError                unsigned int     Error for invalid operation
   fs_accStatus               int              Operations being performed
   fs_states                  int              File system state
   fs_nodeMax                 int              Maximum inode number
   fs_minQuota                int              Minimum inode number
   fs_type                    int              Type of file system
   fs_threshold               char             FSFULL threshold monitoring
   fs_increment               char             FSFULL monitoring increment
   fs_mountstate              char             Mount status
                                               0 - Not mounted
                                               1 - Mounted R/W
                                               2 - Mounted readonly
   fs_msglen                  char             Length of status message
   fs_msg                     char[128]        Status message
   fs_aggrname                char[45]         Aggregate name
   fs_reserved1               char[3]          Reserved
   fs_reserved2               unsigned int[3]  Reserved
   fs_InodeTbl                unsigned int     Size of Inode table
   fs_requests
     high                     unsigned int     High portion of number of file
                                               system requests by applications
     low                      unsigned int     Low portion of number of file
                                               system requests by applications
   fs_reserved3               unsigned int     Reserved
   fs_reserved4               unsigned int     Reserved
   fs_reserved5               unsigned int     Reserved
   fs_diskFormatMajorVersion  unsigned int     Major version of disk format
   fs_diskFormatMinorVersion  unsigned int     Minor version of disk format
   fs_allocLimit_hyper        hyper            Allocation limit for file system
   fs_allocUsage_hyper        hyper            Amount of allocation used
   fs_visQuotaLimit_hyper     hyper            Quota for file system
   fs_visQuotaUsage_hyper     hyper            Amount of quota used
   fs_reserved                char[44]         Reserved


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

Reason_code
  0xEFnnxxxx See z/OS Distributed File Service Messages and Codes
```

Usage notes

- The aggregate must be mounted (or attached).
- For an FS_STATUS, if a size is too large for 32 bits, 0xFFFFFFFF is returned. For an FS_STATUS2, sizes are returned in both the normal fields and the hyper fields.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

List Attached Aggregate Names
List Detailed File System Information

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).

The following fields are internal use only and not intended for application use:
- fs_accError
- fs_accStatus
- fs_type

The fs_states field contains flag 0x00010000, indicating a read/write file system, and flag 0x00030000, indicating a backup file system. All other flags in this field are internal use only and are not intended for application usage.

Examples

Example 1 uses an FS_ID; see Example 2 for an example that uses 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 int  high;
  unsigned int  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;
```

```
struct timeval {
  int          tv_sec;  /* seconds */
  int          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;         /* Flags */
#define         FS_PERFINFO 0x80  /*Performance information in output status*/
  hyper         fs_id;            /*Internal identifier */
  struct timeval fs_cloneTime;    /*Time when this filesys made via
                                    clone or when last recloned */
  struct timeval fs_createTime;   /*Time when this filesys was created */
  struct timeval fs_updateTime;   /*Time when this filesys was last updates*/
  struct timeval fs_accessTime;   /*Time when this filesys was last accessed*/
  unsigned int   fs_allocLimit;  /*Allocation limit for filesys in kilobytes*/
  unsigned int   fs_allocUsage;   /*Amount of allocation used in kilobytes*/
  unsigned int   fs_visQuotaLimit; /*Visible filesystem quota in kilobytes*/
  unsigned int   fs_visQuotaUsage; /*How much quota is used in kilobytes*/
  unsigned int   fs_accError;   /*error to return for incompatible vnode ops */
  int            fs_accStatus;    /*Operations currently being
                                    performed on file system */
  int            fs_states;       /*State bits*/
#define          FS_TYPE_RW 0x10000   /* read/write (ordinary) */
#define          FS_TYPE_BK 0x30000   /* ``.backup */
  int            fs_nodeMax;          /* Maximum inode number used */
  int            fs_minQuota;
  int            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 int fs_reserved2[3];       /* reserved */
  unsigned int 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 */
  unsigned int   fs_reserved3;
  unsigned int   fs_reserved4;
  unsigned int   fs_reserved5;
  int            fs_pad1;
  unsigned int   fs_diskFormatMajorVersion; /* disk format major version */
  unsigned int   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;

    /* file system name to getstatus */
    char              filesystemname[45] = "PLEX.DCEIMGQX.FS";

    struct parmstruct myparmstruct;
    FS_ID             *idp          = &(myparmstruct.fs_id);
    FS_STATUS         *fsp          = &(myparmstruct.fs_status);

    myparmstruct.myparms.opcode = FSOP_GETSTAT_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(FS_ID);
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    memset(idp, 0, sizeof(FS_ID));     /* Ensure reserved fields are 0 */
    memset(fsp, 0, sizeof(FS_STATUS)); /* Ensure reserved fields are 0 */
    memcpy(&myparmstruct.fs_status.fs_eye[0], FS_EYE, 4);
    myparmstruct.fs_status.fs_len = sizeof(FS_STATUS);
    myparmstruct.fs_status.fs_ver = FS_VER_INITIAL;
    memcpy(&myparmstruct.fs_id.fsid_eye, FSID_EYE, 4);
    myparmstruct.fs_id.fsid_len = sizeof(FS_ID);
    myparmstruct.fs_id.fsid_ver = FSID_VER_INITIAL;
    strcpy(myparmstruct.fs_id.fsid_name, filesystemname);

    BPX1PCT("ZFS     ",
            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((const long*) &myparmstruct.fs_status.fs_cloneTime.tv_sec),
             ctime((const long*) &myparmstruct.fs_status.fs_createTime.tv_sec),
             ctime((const long*) &myparmstruct.fs_status.fs_updateTime.tv_sec),
             ctime((const long*) &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);
  }
  return 0;
}
```

The following example uses FS_ID2; see Example 1 for an example that uses FS_ID.

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

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 int  high;
    unsigned int  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;

struct timeval {
    int           tv_sec;  /* seconds */
    int           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;           /* Flags */
#define            FS_PERFINFO 0x80    /* Performance information in
                                          output status */
    hyper          fs_id;              /* Internal identifier */
    struct timeval fs_cloneTime;       /* Time when this filesys made via
                                          clone or when last recloned */
    struct timeval fs_createTime;      /* Time when this filesys
                                          was created */
    struct timeval fs_updateTime;      /* Time when this filesys
                                          was last updated */
    struct timeval fs_accessTime;      /* Time when this filesys
                                          was last accessed */
    unsigned int   fs_allocLimit;      /* Allocation limit for filesys
                                          in kilobytes*/
    unsigned int   fs_allocUsage;      /* Amount of allocation used
                                          in kilobytes*/
    unsigned int   fs_visQuotaLimit;   /* Visible filesystem quota
                                          in kilobytes*/
    unsigned int   fs_visQuotaUsage;   /* How much quota is used in kilobytes*/
    unsigned int   fs_accError;        /* error to return for
                                          incompatible vnode ops */
    int            fs_accStatus;       /* Operations currently being
                                          performed on file system */
    int            fs_states;          /* State bits */
#define            FS_TYPE_RW 0x10000  /* read/write (ordinary) */
#define            FS_TYPE_BK 0x30000  /* ``.backup'' */
    int            fs_nodeMax;         /* Maximum inode number used */
    int            fs_minQuota;
    int            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 int   fs_reserved2[3];    /* reserved */
    unsigned int   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 */
    unsigned int   fs_reserved3;
    unsigned int   fs_reserved4;
    unsigned int   fs_reserved5;
    int            fs_pad1;
    unsigned int   fs_diskFormatMajorVersion; /* disk format major version */
    unsigned int   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;

  /* file system name to getstatus */
  char             filesystemname[45] = "PLEX.DCEIMGQX.FS";

  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((const long*) &myparmstruct.fs_status.fs_cloneTime.tv_sec),
           ctime((const long*) &myparmstruct.fs_status.fs_createTime.tv_sec),
           ctime((const long*) &myparmstruct.fs_status.fs_updateTime.tv_sec),
           ctime((const long*) &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);
  }
  return 0;
}
```

List Systems

Purpose

Retrieves the system names that are part of the zFS XCF group.

Format

```
syscall_parmlist
   opcode           int               174     CFGOP_LSSYS
   parms[0]         int               size of buffer
   parms[1]         int               offset to buffer
   parms[2]         int               offset to bytes returned
   parms[3]         int               0
   parms[4]         int               0
   parms[5]         int               0
   parms[6]         int               0
buffer              char[ ]
bytes_returned      int

Return_value     0 if request successful, -1 if it is not successful

Return_code
  E2BIG D        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 notes

- Reserved fields and undefined flags must be set to binary zeros.
- An array of char[9] fields is returned in buffer. Each element in the array contains a NULL-terminated string with a system name.
- `Bytes_returned / 9` is the number of elements in the array.

Privilege required

None.

Related services

Query sysplex_state

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_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               buffer_success = 0;
  int               bpxrv;
  int               bpxrc;
  int               bpxrs;
  int               i,t;
  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 */

  for(t = 0; t < 1000 && buffer_success == 0; t++)
  {
    if (bpxrv < 0)
    {
      if (bpxrc == E2BIG)
      {
        buflen = myparmstruct.size; /* Get buffer size needed */
        mypsize = sizeof(syscall_parmlist) +
                  buflen +
                  sizeof(myparmstruct.size);

        free(myp);

        myp = (struct parmstruct *)malloc((int)mypsize);
        memset(myp, 0, mypsize);
        myp->myparms.opcode = CFGOP_LSSYS;
        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;
        myp->myparms.parms[4] = 0;
        myp->myparms.parms[5] = 0;
        myp->myparms.parms[6] = 0;

        BPX1PCT("ZFS     ",
                ZFSCALL_CONFIG,     /* Config query operation */
                mypsize,            /* Length of Argument */
                (char *)myp,        /* Pointer to Argument */
                &bpxrv,             /* Pointer to Return_value */
                &bpxrc,             /* Pointer to Return_code */
                &bpxrs);            /* Pointer to Reason_code */

        if( bpxrv != 0 && bpxrc == E2BIG )
          printf("E2BIG: %d times total\n", t++);
        else if( bpxrv == 0 )
        {
          buffer_success = 1;
          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);
        }
        free(myp);
        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);
            }
            free(myp);
            return bpxrc;
          }
        }

        if( t == 1000 )
          printf("Number of failed buffer resizes exceeded.\n");

        free(myp);
        return 0;
      }
```

Query Config Option

Purpose

A set of subcommand calls (configuration operations) that retrieve the current value for a particular configuration setting. Each one returns the configuration setting as a character string in the `co_string` field.

The Format section and Example 1 use the CFGOP_QUERY_ADM_THREADS subcommand. Example 2 shows an example to query the syslevel. The other query subcommands (see Table 17 on page 235) operate in a similar manner.

Format

```
syscall_parmlist
   opcode              int                 180       CFGOP_QUERY_ADM_THREADS
   parms[0]            int                 offset to CFG_OPTION
   parms[1]            int                 offset to system name (optional)
   parms[2]            int                 0
   parms[3]            int                 0
   parms[4]            int                 0
   parms[5]            int                 0
   parms[6]            int                 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_reserved   int[4]              reserved
co_reserved            char[24]            0
systemname             char[9]

Return_value    0 if request is successful, -1 if it is not successful
Return_code
  EBUSY         Aggregate could not be quiesced
  EINTR         ZFS is shutting down
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request
Reason_code
  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.
- The output is the null-terminated string that is returned in `co_string`.

Privilege required

None.

Related services

Set Config Option

Restrictions

None.

Examples

Example 1: The following example shows an API to query admin threads.

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_CONFIG 0x40000006
#define CFGOP_QUERY_ADM_THREADS 180 /* query number of admin threads */

typedef struct syscall_parmlist_t {
  int   opcode;        /* Operation code to perform */
  int   parms[7];      /* Specific to type of operation, */
                       /* provides access to the parms */
                       /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef struct config_option_t {
  char  co_eye[4];             /* Eye catcher */
#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;

int main(int argc, char **argv)
{
  int        bpxrv;
  int        bpxrc;
  int        bpxrs;
  CFG_OPTION *coptr = &(myparmstruct.co);

  /* This next field should only be set if parms[1] is non-zero */

  /* strcpy(myparmstruct.system,"DCEIMGVN"); */ /* set system to query */
  myparmstruct.myparms.opcode = CFGOP_QUERY_ADM_THREADS;
  myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
  myparmstruct.myparms.parms[1] = 0;

  /* Only specify a non-zero offset for the next field (parms[1]) if you are */
  /* z/OS 1.7 and above, and you want to configquery to a different system */

  /* myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + */
  /*                                 sizeof(CFG_OPTION); */

  myparmstruct.myparms.parms[2] = 0;
  myparmstruct.myparms.parms[3] = 0;
  myparmstruct.myparms.parms[4] = 0;
  myparmstruct.myparms.parms[5] = 0;
  myparmstruct.myparms.parms[6] = 0;

  memset(coptr, 0, sizeof(CFG_OPTION));
  memcpy(coptr->co_eye, CFGO_EYE, 4);
  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 -adm_threads, "
           "BPXRV = %d BPXRC = %d BPXRS = %x\n",
           bpxrv, bpxrc, bpxrs);
    return bpxrc;
  }
  else
  {
    printf("Config query -adm_threads = %s\n", myparmstruct.co.co_string);
  }
  return 0;
}
```

Example 2: The following example shows an API to query the syslevel.

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
#include <string.h>

#define ZFSCALL_CONFIG 0x40000006
#define CFGOP_QUERY_SYSLEVEL 238 /* Query Config option - syslevel */

/* Not in a sysplex shared file system environment */
#define NO_SYSPLEX_SUPPORT 0
/* Admin level sysplex shared file system environment */
#define SYSPLEX_ADMIN_LEVEL 1
/* File level sysplex shared file system environment */
#define SYSPLEX_FILE_LEVEL 2
/* Sysplex-aware on a File system basis */
#define SYSPLEX_FILESYS_LEVEL 3

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 vaalues */
  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);

  char       *version,
             *service,
             *created,
             *sysplex,
             *interface,
             *rwshare_default,
             *rest;

  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 */
    /* format is */
    /* "OSlevel\nServicelevel\ncreatetimestamp\" +                  */
    /* "nsysplex_state\ninterface_level\nrwshare_default\0"         */

    version = myparmstruct.co.co_string;
    service = strchr(version, '\n'); /* find the end of the    */
                                     /* version (for 2nd line) */
    *service = '\0';     /* ensure end of string for version string */
    service++;           /* increment to next field (service) */
    created = strchr(service, '\n'); /* find the end of the   */
                                     /*service (for 2nd line) */
    *created = '\0';     /* ensure end of string for service string */
    created++;           /* increment to next field (creation) */

    sysplex = strchr(created, '\n'); /* find the end of the */
                                     /* creation timestamp */
    *sysplex = '\0';     /* ensure end of string for creation string */
    sysplex++;           /* increment to next field (sysplex_state) */

    interface = strchr(sysplex, '\n');  /* find end of the sysplex_state */
    *interface = '\0';   /* ensure end of string for sysplex_state */
    interface++;         /* increment to next field (interface level) */

    sysplex_level = atoi(sysplex);
    if (sysplex_level == NO_SYSPLEX_SUPPORT)
    {
      printf("zFS kernel: z/OS File System\nVersion %s "
             "Service Level %s.\n Created on %s.\n",
             version, service, created);
    }
    else
    {
      char buffer[80];

      /* find the end of the interface */
      rwshare_default = strchr(interface, '\n');
      if (rwshare_default != NULL)
      {
        *rwshare_default = '\0';
        rwshare_default++;
      }
      if (sysplex_level == SYSPLEX_ADMIN_LEVEL)
        sprintf(buffer, "sysplex(admin-only) interface(%s)", interface);
      else /* if sysplex_level is SYSPLEX_FILE_LEVEL */
      {
        if (sysplex_level == SYSPLEX_FILE_LEVEL)
          sprintf(buffer, "sysplex(file) interface(%s)", interface);
        else
        {  /* if sysplex_level is SYSPLEX_FILESYS_LEVEL */
          if (sysplex_level == SYSPLEX_FILESYS_LEVEL)
          {
            /* find the end of rwshare_default */
            rest = strchr(rwshare_default, '\n');
            if (rest != NULL)
              *rest = '\0'; /*ensure that rwshare_default is null terminated*/
            sprintf(buffer, "sysplex(filesys,%s) interface(%s)",
                    rwshare_default, interface);
          }
          else
            sprintf(buffer, "sysplex(%s) interface(%s)", sysplex, interface);
        }
      }
      printf("zFS kernel: z/OS File System\nVersion "
             "%s Service Level %s.\nCreated on %s.\n%s\n",
             version, service, created, buffer);
    }
  }
  return 0;
}
```

Quiesce Aggregate

Purpose

An aggregate operation that quiesces a compatibility mode aggregate. It quiesces activity on the aggregate and its file system.

Format

```
syscall_parmlist
   opcode               132             AGOP_QUIESCE_PARMDATA
   parms[0]             int             offset to AGGR_ID
   parms[1]             int             offset to handle returned by quiesce
   parms[2]             int             0
   parms[3]             int             0
   parms[4]             int             0
   parms[5]             int             0
   parms[6]             int             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          int

Return_value    0 if request is successful, -1 if it is not successful

Return_code
  EBUSY         Aggregate could not be quiesced
  EINTR         ZFS is shutting down
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request
Reason_code
  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Quiesce Aggregate is used to suspend activity on an aggregate. All activity on the file system contained in the aggregate that is mounted is also suspended. This subcommand is typically used before backing up an aggregate. The aggregate must be attached to be quiesced. The quiesce operation returns a quiesce handle that must be supplied on the unquiesce call.
- 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

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

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;
    int             quiesce_handle;
};

int main(int argc, char **argv)
{
    int              bpxrv;
    int              bpxrc;
    int              bpxrs;
    char             aggrname[45] = "PLEX.DCEIMGQX.FS";
    int              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;

    /* Ensure reserved fields are 0 */
    memset(&myparmstruct.aggr_id, 0, sizeof(AGGR_ID));
    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 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;
    }
```

Reset Backup Flag

Purpose

Used by backup programs to reset the backup bit after completion of a backup. The backup program is expected to quiesce the aggregate and save the quiesce handle before beginning the backup. After completing the backup, the backup bit should be reset before unquiescing the aggregate.

Format

```
syscall_parmlist
   opcode               int           157 AGOP_RESETFLAG_PARMDATA
   parms[0]             int           offset to AGGR_ID
   parms[1]             int           quiesce handle
   parms[2]             int           0
   parms[3]             int           0
   parms[4]             int           0
   parms[5]             int           0
   parms[6]             int           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
  EINVAL   Invalid input parameters
  ENOENT   Aggregate not found
  ENOSYS   Aggregate not locally owned
  EBUSY    Aggregate is growing
  EMVSERR  Internal error using an osi service

Reason_code
  0xEFnnxxxx   See z/OS Distributed File Service Messages and Codes
  EINVAL       Invalid parameters

Reason_code
  0xEFnnxxxx   See z/OS Distributed File Service Messages and Codes
```

Usage notes

- The backup bit must be reset while the aggregate is still quiesced for backup.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

Quiesce Aggregate
Unquiesce Aggregate

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_RESETFLAG_PARMDATA 157

typedef struct syscall_parmlist_t
{
    int  opcode;              /* Operation code to perform */
    int  parms[7];            /* Specific to type of operation, */
                              /* provides access to the parms */
                              /* parms[2]-parms[6] are currently unused*/
} syscall_parmlist;

#define ZFS_MAX_AGGRNAME 44

typedef struct aggr_id_t
{
    char aid_eye[4];                    /* Eye Catcher */
#define  AID_EYE "AGID"
    char aid_len;                       /* Length of this structure */
    char aid_ver;                       /* Version */
#define  AID_VER_INITIAL 1              /* Initial version */
    char aid_name[ZFS_MAX_AGGRNAME+1];  /* aggr name, null terminated */
    char aid_reserved[33];              /* Reserved for the future */
} AGGR_ID;

struct parmstruct {
    syscall_parmlist myparms;
    AGGR_ID          aggr_id;
};

int main(int argc, char **argv)
{
    int              bpxrv;
    int              bpxrc;
    int              bpxrs;

    /*Aggregate name to attach, aggregate must
      be quiesced for this API to run successfully */
    char             aggrname[45]       = "PLEX.DCEIMGQX.FS";

    struct parmstruct myparmstruct;
    AGGR_ID          *idp           = &(myparmstruct.aggr_id);

    /* This is the handle returned by zFS on a quiesce aggregate   */
    /* Ensure that the quiesce_handle is set to the value returned */
    /* by the quiesce                                              */
    int              quiesce_handle = 1;

    myparmstruct.myparms.opcode   = AGOP_RESETFLAG_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = quiesce_handle;
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;
    memset(idp, 0, sizeof(AGGR_ID));      /* Ensure reserved fields are 0 */

    memcpy(&myparmstruct.aggr_id.aid_eye, AID_EYE, 4);
    myparmstruct.aggr_id.aid_len = sizeof(AGGR_ID);
    myparmstruct.aggr_id.aid_ver = AID_VER_INITIAL;
    strcpy(myparmstruct.aggr_id.aid_name, aggrname);
```

```
        BPX1PCT("ZFS     ",
                ZFSCALL_AGGR,                /* Aggregate operation */
                sizeof(myparmstruct),        /* Length of Argument */
                (char *)&myparmstruct,       /* Pointer to Argument */
                &bpxrv,                      /* Pointer to Return_value */
                &bpxrc,                      /* Pointer to Return_code */
                &bpxrs);                     /* Pointer to Reason_code */

        if (bpxrv < 0)
        {
            printf("Error resetting backup flag for aggregate %s\n", aggrname);
            printf("BPXRV = %d BPXRC = %d BPXRS = %x\n", bpxrv, bpxrc, bpxrs);
            return bpxrc;
        }
        else /* Return from reset was successful */
            printf("Successfully reset backup flag for aggregate %s\n", aggrname);
        return 0;
    }
```

Set Auditfid

Purpose

An aggregate operation that sets the current value of the auditfid. The aggregate whose auditfid is to be changed must be attached.

Format

```
syscall_parmlist
   opcode         int         149       AGOP_SETAUDITFID_PARMDATA
   parms[0]       int         offset to AGGR_ID
   parms[1]       int         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]       int         0
   parms[3]       int         0
   parms[4]       int         0
   parms[5]       int         0
   parms[6]       int         0
AGGR_ID
   aid_eye        char[4]     "AGID"
   aid_len        char        sizeof(AGGR_ID)
   aid_ver        char        1
   aid_name       char[45]    "OMVS.PRV.AGGR001.LDS0001"
   aid_reserved   char[33]    0

Return_value    0 if request is successful, -1 if it is not successful

Return_code
  EBUSY         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 notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must be UID=0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

List Aggregate Status (Version 2)

Restrictions

The aggregate cannot be attached as read-only. The aggregate cannot be quiesced. The aggregate cannot be in the process of being moved by zFS.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_AGGR 0x40000005
#define AGOP_SETAUDITFID_PARMDATA 149  /* Set or reset auditfid */

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;
    struct parmstruct myparmstruct;

    char aggrname[45] = "PLEX.DCEIMGQX.FS"; /* aggregate name to set auditfid*/
    AGGR_ID *idp      = &(myparmstruct.aggr_id);

    myparmstruct.myparms.opcode = AGOP_SETAUDITFID_PARMDATA;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);

    /* Configure options by setting myparmstruct.myparms.parms[1] to:   */
    /* 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[1] = 1;

    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;

    /* Ensure reserved fields are 0 */
    memset(&myparmstruct.aggr_id, 0, sizeof(AGGR_ID));
    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 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

A set of subcommand calls (that are configuration operations) that set the current value for a particular configuration setting. Each one sets the configuration setting from input specified as a character string.

The following Format and Example use the CFGOP_ADM_THREADS subcommand. The other set subcommands (see Table 17 on page 235) operate similarly. That is, each sets the configuration setting from the character string in the `co_string` field.

Format

```
syscall_parmlist
   opcode                int          150        CFGOP_ADM_THREADS
   parms[0]              int          offset to CFG_OPTION
   parms[1]              int          offset to system name (optional)
   parms[2]              int          0
   parms[3]              int          0
   parms[4]              int          0
   parms[5]              int          0
   parms[6]              int          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          4 (reserved)
   co_reserved           char[24]     0
systemname               char[9]

Return_value    0 if request is successful, -1 if it is not successful

Return_code
  EBUSY         Aggregate could not be quiesced
  EINTR         ZFS is shutting down
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request
Reason_code
  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.
- Specify the new value as a null terminated string in `co_string`.

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

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

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;
  myparmstruct.myparms.parms[5] = 0;
  myparmstruct.myparms.parms[6] = 0;

  memset(coptr, 0, sizeof(CFG_OPTION));
  memcpy(coptr->co_eye, CFGO_EYE, 4);
  coptr->co_ver = CO_VER_INITIAL;
  coptr->co_len = (int)sizeof(CFG_OPTION);
  strcpy(coptr->co_string, new_adm_threads);/*set new adm_thread value*/

  BPX1PCT("ZFS     ",
          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;
}
```

Statistics Directory Cache Information

Purpose

Returns directory cache counters, including the number of requests, hits and discards from the directory cache.

Note: As of z/OS V1R13, this subcommand is no longer used. All output from a call to statistics directory cache information will be zeros.

Format

```
syscall_parmlist
   opcode              int              249             STATOP_DIR_CACHE
   parms[0]            int              offset to STAT_API
   parms[1]            int              offset of output following STAT_API
   parms[2]            int              offset to system name (optional)
   parms[3]            int              0
   parms[4]            int              0
   parms[5]            int              0
   parms[6]            int              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 int     high order 32 bits since epoch
   posix_time_low      unsigned int     low order 32 bits since epoch
   posix_useconds      unsigned int     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
systemname             char[9]

Return_value     0 if request is successful, -1 if it is not successful
Return_code
  EINTR          zFS is shutting down
  EINVAL         Invalid parameter list
  EMVSERR        Internal error occurred
  E2BIG          Information too big for buffer supplied
Reason_code
  0xEFnnxxxx     See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

Statistics Vnode Cache Information
Statistics Metadata Cache Information

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
#include <string.h>
#include <time.h>

#define ZFSCALL_STATS 0x40000007
#define STATOP_DIR_CACHE 249 /* Directory cache stats */
#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 int      high; /* unsigned int  reserved */
  unsigned int      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 {
  unsigned int           posix_time_high; /* high order 32 bits since epoc */
  unsigned int           posix_time_low;  /* low order 32 bits since epoch */
  unsigned int           posix_usecs;     /* microseconds */
  int                    pad1;
} RESET_TIME;

/*********************************************************************/
/* The following structure is the api query control block */
/* It is used for all api query commands */
/*********************************************************************/
typedef struct stat_api_t
{
#define SA_EYE "STAP"
  char             sa_eye[4];     /* 4 byte identifier must be */
  int              sa_len;        /* length of the buffer to put data into*/
  /* this buffer area follows this struct*/
  int              sa_ver;        /* the version number currently always 1*/
#define              SA_VER_INITIAL 0x01
  char             sa_flags;      /* flags field must be x00 or x80,
                                     x80 means reset statistics*/
#define              SA_RESET 0x80
  char             sa_fill[3];    /* spare bytes */
  int              sa_reserve[4]; /* Reserved */
  struct reset_time reset_time_info;
} STAT_API;

struct parmstruct {
  syscall_parmlist myparms;
  STAT_API         myapi;
  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;
  int      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));
  memcpy(stapptr->sa_eye, SA_EYE, 4);
  stapptr->sa_ver = SA_VER_INITIAL;
  stapptr->sa_len = (int)sizeof(API_DIR_STATS);

  /* This next field should only be set if parms[2] is non-zero */
  /* strcpy(myparmstruct.systemname,"DCEIMGVQ"); */
  BPX1PCT("ZFS     ",
          ZFSCALL_STATS,           /* Perf statistics operation */
          sizeof(myparmstruct),    /* Length of Argument */
          (char *)&myparmstruct,   /* Pointer to Argument */
          &bpxrv,                  /* Pointer to Return_value */
          &bpxrc,                  /* Pointer to Return_code */
          &bpxrs);                 /* Pointer to Reason_code */

  if (bpxrv < 0)
  {
    printf("Error querying directory cache, "
           "BPXRV = %d BPXRC = %d BPXRS = %x\n",
            bpxrv, bpxrc, bpxrs);
    return bpxrc;
  }
  else
  {
    printf("\n%50s\n", "Directory Backing Caching Statistics");
    printf(" \n");
    printf("Buffers    (K bytes)  Requests     Hits    Ratio   Discards  \n");
    printf("---------- --------- ---------- ---------- ------ ---------- \n");

    if( myparmstruct.mystats.ad_requests.low == 0 )
      temp_ratio = 0;
    else
      temp_ratio = ((double)myparmstruct.mystats.ad_hits.low) /
                   myparmstruct.mystats.ad_requests.low;

    temp_ratio *= 100.0;
    CONVERT_RATIO_TO_INTS(temp_ratio, whole, decimal);

    decimal = decimal / 100; /* Just want tenths */
    printf("%10u %9u %10u %10u %3u.%1.1u%% %10u\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

Displays information about the number of reads and writes (I/Os) and the amount of data in bytes that are transferred for each aggregate.

Format

```
syscall_parmlist
   opcode                          int             244              STATOP_IOBYAGGR
   parms[0]                        int             offset to STAT_API
   parms[1]                        int             offset of output following STAT_API
   parms[2]                        int             offset to system name (optional)
   parms[3]                        int             0
   parms[4]                        int             0
   parms[5]                        int             0
   parms[6]                        int             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]         0x80 - Reset statistics
   sa_reserve                      int[3]          Reserved
   posix_time_high                 unsigned int    High order 32 bits since epoch
   posix_time_low                  unsigned int    Low order 32 bits since epoch
   posix_useconds                  unsigned int    Microseconds
IO_REPORT2_2_GRAND_TOTALS
   io_count                        int             Count of IO_REPORT2 lines
   grand_total_reads               unsigned int    Total reads
   grand_total_writes              unsigned int    Total writes
   grand_total_read_bytes          unsigned int    Total bytes read (in kilobytes)
   grand_total_write_bytes         unsigned int    Total bytes written (in kilobytes)
   grand_total_devices             unsigned int    Total number of aggregates
   total_number_waits_for_io       unsigned int    Total number of waits for I/O
   average_wait_time_for_io_whole  unsigned int    Average wait time (whole number),
                                                   average wait time in milliseconds
   average_wait_time_for_io_decimal  unsigned int  Average wait time (decimal part)
                                                   decimal part is in thousanths
                                                   3 means .003 and 300 means .3
```

```
IO_REPORT2[io_count]
   volser                        char[8]             DASD volser where aggregate resides
   pavios                        unsigned int        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 int        Count of reads for this aggregate
   temp_read_bytes               unsigned int        Bytes read for this aggregate (in kilobytes)
   temp_writes                   unsigned int        Count of writes for this aggregate
   temp_write_bytes              unsigned int        Bytes written for this aggregate (in kilobytes)
   allocation_dsname             char[84]            Data set name of aggregate
--or--
IO_REPORT2_GRAND_TOTALS2
   io_count                      int                 Count of IO_REPORT2 lines
   grand_total_reads             unsigned long long  Total reads
   grand_total_writes            unsigned long long  Total writes
   grand_total_read_bytes        unsigned long long  Total bytes read (in kilobytes)
   grand_total_write_bytes       unsigned long long  Total bytes written (in kilobytes)
   grand_total_devices           unsigned long long  Total number of aggregates
   total_number_waits_for_io     unsigned long long  Total number of waits for I/O
   average_wait_time_for_io_whole    unsigned int    Average wait time (whole number),
                                                     average wait time in milliseconds
   average_wait_time_for_io_decimal  unsigned int    Average wait time (decimal part)
                                                     decimal part is in thousanths
                                                     3 means .003 and 300 means .3
IO_REPORT2_2[io_count]
   volser                        char[8]             DASD volser where aggregate resides
   pavios                        unsigned int        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 long  Count of reads for this aggregate
   temp_read_bytes               unsigned long long  Bytes read for this aggregate (in kilobytes)
   temp_writes                   unsigned long long  Count of writes for this aggregate
   temp_write_bytes              unsigned long 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 notes

- Reserved fields and undefined flags must be set to binary zeros.
- When `sa_supported_ver` is 0 or 1, output consists of IO_REPORT2_GRAND_TOTALS and IO_REPORT2. When `sa_supported_ver` is 2, output consists of IO_REPORT2_GRAND_TOTALS2 and IO_REPORT2_2.

Privilege required

None.

Related services

Statistics iobydasd Information
Statistics iocounts Information

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.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 {
  unsigned int           posix_time_high; /* high order 32 bits since epoc */
  unsigned int           posix_time_low;  /* low order 32 bits since epoch */
  unsigned int           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_2 0x02
#define            SA_VER_INIT 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_supported_ver; /* version of data returned */
  int              sa_reserve[3];    /* Reserved */
  struct reset_time reset_time_info;
} STAT_API;

typedef struct io_report2_2_t {
  char volser[8];
  unsigned int pavios;
  char read_ind[4];
  unsigned long long int temp_reads;
  unsigned long long int temp_read_bytes;
  unsigned long long int temp_writes;
  unsigned long long int temp_write_bytes;
  char  allocation_dsname[84];
  char reserved[4];
} IO_REPORT2_2;

typedef struct io_report2_grand_totals_2_t {
  int  io_count; /* number IO_REPORT2 structs in buffer */
  int pad;
  unsigned long long int grand_total_reads;       /* Total # reads      */
  unsigned long long int grand_total_writes;      /* Total # writes     */
  unsigned long long int grand_total_read_bytes;  /* Total bytes read   */
  unsigned long long int grand_total_write_bytes; /* Total bytes written*/
  unsigned long long int grand_total_devices;     /* total # aggregates */
  unsigned long long int total_number_waits_for_io;
  unsigned int average_wait_time_for_io_whole;
  unsigned int average_wait_time_for_io_decimal;
} IO_REPORT2_GRAND_TOTALS_2;

/* Version 1 Output structures */
typedef struct io_report2_t {
  char             volser[8];
  unsigned int     pavios;
  char             read_ind[4];
  unsigned int     temp_reads;
  unsigned int     temp_read_bytes;
  unsigned int     temp_writes;
  unsigned int     temp_write_bytes;
  char             allocation_dsname[84];
} IO_REPORT2;

typedef struct io_report2_grand_totals_t {
  int              io_count;                         /* number IO_REPORT2
                                                         structs in buffer */
  unsigned int     grand_total_reads;                /* Total # reads */
  unsigned int     grand_total_writes;               /* Total # writes */
  unsigned int     grand_total_read_bytes;           /* Total bytes read */
  unsigned int     grand_total_write_bytes;          /* Total bytes written*/
  unsigned int     grand_total_devices;              /* total # aggregates */
  unsigned int     total_number_waits_for_io;
  unsigned int     average_wait_time_for_io_whole;   /* in milliseconds */
  unsigned int     average_wait_time_for_io_decimal; /* in thousandths  */
                                                     /* of milliseconds */
                                                     /* for example,    */
                                                     /*3 means .003 and
                                                        300 means .3 */
} IO_REPORT2_GRAND_TOTALS;

struct parmstruct {
  syscall_parmlist myparms;
  STAT_API        myapi;

  /* output buffer IO_REPORT2_GRAND_TOTALS_2 + multiple IO_REPORT2_2s */
  char            systemname[9];
} myparmstruct;

int print_iobyaggr_version1(IO_REPORT2_GRAND_TOTALS *stgt,
                            IO_REPORT2              *str2);
int print_iobyaggr_version2(IO_REPORT2_GRAND_TOTALS_2 *stgt,
                            IO_REPORT2_2              *str2);

int main(int argc, char **argv)
{
  int                       buffer_success = 0;
  int                       bpxrv;
  int                       bpxrc;
  int                       bpxrs;
  int                       i,t;
  IO_REPORT2_GRAND_TOTALS_2 *stgt;
  IO_REPORT2_2              *str2;
  char                      *stsy;
  char                      buf[33];
  struct parmstruct         *myp     = &myparmstruct;
  int                       mypsize;
  int                       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);

  /* 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[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_2;
  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 */

  for(t = 0; t < 1000 && buffer_success == 0; t++)
  {
    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);

        free(myp);

        myp     = (struct parmstruct *)malloc((int)mypsize);
        memset(myp, 0, mypsize);

        printf("Need buffer size of %d, for a total of %d\n\n\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_2;
        stapptr->sa_len = buflen;
        stgt = (IO_REPORT2_GRAND_TOTALS_2 * )((char *)myp +
                                              sizeof(syscall_parmlist) +
                                              sizeof(STAT_API));

        str2 = (IO_REPORT2_2*) ((char*) stgt +
                                sizeof(IO_REPORT2_GRAND_TOTALS_2));
        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 */

          if( bpxrv != 0 && bpxrc == E2BIG )
            printf("E2BIG: %d times total\n", t++);
          else if( bpxrv == 0 )
          {
            buffer_success = 1;

            if (stapptr->sa_supported_ver == SA_VER_INIT)
            {
              IO_REPORT2_GRAND_TOTALS *stgt_v1;
              IO_REPORT2 *str2_v1;
              stgt_v1 = (IO_REPORT2_GRAND_TOTALS * )((char *)myp +
                     sizeof(syscall_parmlist) +
                     sizeof(STAT_API));

              str2_v1 = (IO_REPORT2 * ) ((char*) stgt +
                     sizeof(IO_REPORT2_GRAND_TOTALS));
              print_iobyaggr_version1(stgt_v1,str2_v1);
            }
            else
              print_iobyaggr_version2(stgt, str2);

            unsigned int ptl = stapptr->reset_time_info.posix_time_low;
            if (0 == ctime_r((time_t * ) & ptl, 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", bpxrv, bpxrc, bpxrs);
        }
        free(myp);
        return bpxrc;
      }
    }

    if( t == 1000 )
      printf("Number of failed buffer resizes exceeded.\n");

    free(myp);
    return 0;
  }

  int print_iobyaggr_version2(IO_REPORT2_GRAND_TOTALS_2 *stgt,
                              IO_REPORT2_2              *str2)
  {
      int i;
      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");
      printf("------ --- ---- ---------- ---------- "
             "---------- ---------- ------------\n");

      for (i = 0; i < stgt->io_count; i++, str2++)
      {
        printf("%6.6s %3u %s %10llu %10llu %10llu %10llu  %-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("%6llu         %10llu %10llu %10llu %10llu  %-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: %10u\n",
             stgt->total_number_waits_for_io);
      printf("Average I/O wait time:         %9u.%3.3u (msecs)\n",
             stgt->average_wait_time_for_io_whole,
             stgt->average_wait_time_for_io_decimal);
      printf("\n");
      return 1;
  }

  int print_iobyaggr_version1(IO_REPORT2_GRAND_TOTALS *stgt,
                              IO_REPORT2              *str2)
  {
      int i;
      printf("Version 1 output is being displayed\n");

      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");
      printf("------ --- ---- ---------- ---------- "
             "---------- ---------- ------------\n");

      for (i = 0; i < stgt->io_count; i++, str2++) {
          printf("%6.6s %3u %s %10u %10u %10u %10u  %-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("%6u         %10u %10u %10u %10u  %-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: %10u\n",
             stgt->total_number_waits_for_io);
      printf("Average I/O wait time:         %9u.%3.3u (msecs)\n",
             stgt->average_wait_time_for_io_whole,
             stgt->average_wait_time_for_io_decimal);
      printf("\n");
  }
```

Statistics iobydasd Information

Purpose

Displays information about the number of reads and writes and the number of bytes transferred for each DASD volume. The number of I/Os and the amount of data transferred is determined on a DASD basis.

Format

```
syscall_parmlist
   opcode                    int              245             STATOP_IOBYDASD
   parms[0]                  int              offset to STAT_API
   parms[1]                  int              offset of output following STAT_API
   parms[2]                  int              offset to system name (optional)
   parms[3]                  int              0
   parms[4]                  int              0
   parms[5]                  int              0
   parms[6]                  int              0
STAT_API
   sa_eye                    char[4]          "STAP"
   sa_len                    int              length of buffer that follows STAT_API
   sa_ver                    int              1 or 2
   sa_flags                  char[1]          0x00
     SA_RESET                0x80             Reset statistics
   sa_fill                   char[3]          0
   sa_supported_ver          int              version of data returned
   sa_reserve                int[3]           0
   posix_time_high           unsigned int     high order 32 bits since epoch
   posix_time_low            unsigned int     low order 32 bits since epoch
   posix_useconds            unsigned int     microseconds
   pad1                      int
                             int
API_IOBYDASD_HDR
   number_of_lines           int              count of API_IOBYDASD_DATA lines
   pad                       int              0
   grand_total_waits         hyper            total waits
   average_wait_time_whole   int              average wait time (whole number)
                                              average wait time in milliseconds
   average_wait_time_decimal int              average wait time (decimal part)
                                              decimal part is in thousanths
                                              3 means .003 and 300 means .3
API_IOBYDASD_DATA[number_of_lines]
   spare                    int               0
   volser                   char[6]           DASD volser
   filler                   char[2]           reserved
   pavios                   unsigned int      max number of concurrent I/Os zFS will issue
                                              for this DASD
   reads                    unsigned int      count of reads for this DASD
   read_bytes               unsigned int      bytes read for this DASD (in kilobytes)
   writes                   unsigned int      count of writes for this DASD
   write_bytes              unsigned int      bytes written for this DASD (in kilobytes)
   waits                    unsigned int      waits
   avg_wait_whole           int               average wait time (whole number)
                                              average wait time in milliseconds
   avg_wait_decimal         int               average wait time (decimal part)
                                              decimal part is in thousanths
                                              3 means .003 and 300 means .3
```

```
--or--
API_IOBYDASD_DATA2[number_of_lines]
   spare                  int                       0
   volser                 char[6]                   DASD volser
   filler                 char[2]                   reserved
   unsigned int           unsigned long long int    max number of concurrent I/Os zFS
                                                    will issue for this DASD
   reads                  unsigned long long int    count of reads for this DASD
   read_bytes             unsigned long long int    bytes read for this DASD (in kilobytes)
   writes                 unsigned long long int    count of writes for this DASD
   write_bytes            unsigned long long int    bytes written for this DASD (in kilobytes)
   waits                  unsigned long long int    waits
   avg_wait_whole         int                       average wait time (whole number)
                                                    average wait time in milliseconds
   avg_wait_decimal       int                       average wait time (decimal part)
                                                    decimal part is in thousanths
                                                    3 means .003 and 300 means .3
systemname                char[9]

Return_value     0 if request is successful, -1 if it is not successful

Return_code
  EINTR          zFS is shutting down
  EINVAL         Invalid parameter list
  EMVSERR        Internal error occurred
  E2BIG          Information too big for buffer supplied
Reason_code
  0xEFnnxxxx     See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.
- When sa_supported_ver is 0 or 1, the output consists of API_IOBYDASD_HDR and API_IOBYDASD_DATA. When sa_supported_ver is 2, the output consists of API_IOBYDADD_HDR and API_IOBYDASD_DATA2.

Privilege required

None.

Related services

Statistics iobyaggr Information
Statistics iocounts Information

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_STATS   0x40000007
#define STATOP_IOBYDASD 245     /* Performance API queries */
#define E2BIG           145
#define ENOMEM          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;

typedef struct reset_time {
  unsigned int           posix_time_high; /* high order 32 bits since epoc */
  unsigned int           posix_time_low;  /* low order 32 bits since epoch */
  unsigned int           posix_usecs;     /* microseconds */
  int                    pad1;
} RESET_TIME;

typedef struct hyper_t {
  unsigned int high; /* unsigned int reserved */
  unsigned int 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_2 0x02
#define            SA_VER_INIT 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_supported_ver; /* version of data returned */
  int              sa_reserve[3];    /* Reserved */
  struct reset_time reset_time_info;
} STAT_API;

typedef struct api_iobydasd_hdr
{
  int              number_of_lines;
  int              pad;
  hyper            grand_total_waits;
  int              avg_wait_time_whole;   /* in milliseconds */
  int              avg_wait_time_decimal; /* in thousandths */
                                          /* of milliseconds */
                                          /* for example, 3 means .003 */
                                          /* and 300 means .3          */
} API_IOBYDASD_HDR;

typedef struct api_iobydasd_data_2
{
  int                    spare;
  char                   volser[6];
  char                   filler[2];
  unsigned int           pavios;
  unsigned long long int reads;
  unsigned long long int read_bytes;
  unsigned long long int writes;
  unsigned long long int write_bytes;
  unsigned long long int waits;
  int                    avg_wait_whole;
  int                    avg_wait_decimal;
} API_IOBYDASD_DATA_2;
```

```
/* Version 1 output structure */
typedef struct api_iobydasd_data
{
  int               spare;
  char              volser[6];
  char              filler[2];
  unsigned int      pavios;
  unsigned int      reads;
  unsigned int      read_bytes;
  unsigned int      writes;
  unsigned int      write_bytes;
  unsigned int      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_DATA_2s */
  char             systemname[9];
} myparmstruct;

int print_iobydasd_version1(API_IOBYDASD_HDR* stdh,
                            API_IOBYDASD_DATA *stdd);
int print_iobydasd_version2(API_IOBYDASD_HDR* stdh,
                            API_IOBYDASD_DATA_2 *stdd);

int main(int argc, char **argv)
{
  int                 buffer_success = 0;
  int                 bpxrv;
  int                 bpxrc;
  int                 bpxrs;
  int                 i,t;
  API_IOBYDASD_HDR    *stdh;
  API_IOBYDASD_DATA_2 *stdd;
  char                *stsy;
  char                buf[33];
  struct parmstruct   *myp     = &myparmstruct;
  int                 mypsize;
  int                 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_2;
  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 */

  for(t = 0; t < 1000 && buffer_success == 0; t++)
  {
    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);

        free(myp);
        myp = (struct parmstruct *)malloc((int)mypsize);
        memset(myp, 0, mypsize);

        printf("Need buffer size of %d, for a total of %d\n\n",
               buflen, mypsize);
        myp->myparms.opcode = STATOP_IOBYDASD;
        myp->myparms.parms[0] = sizeof(syscall_parmlist);
        myp->myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
        myp->myparms.parms[2] = 0;

        /* Only specify a non-zero offset for the next field (parms[2]) if */
        /* you are running z/OS 1.7 and above, and you want to query the   */
        /* iobydasd statistics of a different system than this one         */
        /* myp->myparms.parms[2] = sizeof(syscall_parmlist)                */
        /*                       + sizeof(STAT_API) + buflen;              */

        myp->myparms.parms[3] = 0;
        myp->myparms.parms[4] = 0;
        myp->myparms.parms[5] = 0;
        myp->myparms.parms[6] = 0;

        stapptr = (STAT_API * )((char *)myp + sizeof(syscall_parmlist));
        memcpy(stapptr->sa_eye, SA_EYE, 4);
        stapptr->sa_ver = SA_VER_2;
        stapptr->sa_len = buflen;
        stdh = (API_IOBYDASD_HDR * )((char *)myp +
               sizeof(syscall_parmlist) + sizeof(STAT_API));
        stdd = (API_IOBYDASD_DATA_2*)((char*)stdh + sizeof(API_IOBYDASD_HDR));
        stsy = (char *)((char *)myp + sizeof(syscall_parmlist) +
               sizeof(STAT_API) + buflen);

        /* This next field should only be set if parms[2] is non-zero */
        /* strcpy(stsy,"DCEIMGVQ"); */

        BPX1PCT("ZFS     ",
                ZFSCALL_STATS,      /* Perf stats 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 && bpxrc == E2BIG )
          printf("E2BIG: %d times total\n", t++);
        else if( bpxrv == 0 )
        {
          buffer_success = 1;

          if( stapptr->sa_supported_ver == SA_VER_INIT )
          {
            API_IOBYDASD_DATA *stdd_v1;
            stdd_v1 = (API_IOBYDASD_DATA * )((char *)stdh +
                                             sizeof(API_IOBYDASD_HDR));
            print_iobydasd_version1(stdh,stdd_v1);
          }
          else
            print_iobydasd_version2(stdh,stdd);


          unsigned int ptl = stapptr->reset_time_info.posix_time_low;
          if (0 == ctime_r((time_t * ) & ptl, 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);
      }
      free(myp);
      return bpxrc;
    }
  }
  if( t == 1000 )
    printf("Number of failed buffer resizes exceeded.\n");

  free(myp);
  return 0;
}
```

```
int print_iobydasd_version2(API_IOBYDASD_HDR* stdh,
                            API_IOBYDASD_DATA_2 *stdd)
{
  int i;
  printf("%40czFS I/O by Currently Attached DASD/VOLs\n", ' ');
  printf("\n");
  printf("DASD   PAV\n");
  printf("VOLSER IOs Reads                K bytes              "
         "Writes               ");
  printf("K bytes              Waits                Average Wait\n");
  printf("------ --- -------------------- -------------------- "
         "-------------------- ");
  printf("-------------------- -------------------- ------------\n");

  for (i = 0; i < stdh->number_of_lines; i++, stdd++)
  {
    printf("%6.6s %3u %20llu %20llu %20llu %20llu %20llu %6u.%3.3u\n",
           stdd->volser,
           stdd->pavios,
           stdd->reads,
           stdd->read_bytes,
           stdd->writes,
           stdd->write_bytes,
           stdd->waits,
           stdd->avg_wait_whole,
           stdd->avg_wait_decimal);
  }
  printf("\n");
  printf("Total number of waits for I/O: %u,,%u\n",
         stdh->grand_total_waits.high, stdh->grand_total_waits.low);
  printf("Average I/O wait time:         %9u.%3.3u (msecs)\n",
         stdh->avg_wait_time_whole,
         stdh->avg_wait_time_decimal);
  printf("\n");

  return 1;
}

int print_iobydasd_version1(API_IOBYDASD_HDR* stdh,
                            API_IOBYDASD_DATA *stdd)
{
  int i;
  printf("Version 1 output is being displayed\n\n");
  printf("%15c  zFS I/O by Currently Attached DASD/VOLs\n",' ');
  printf("\n");
  printf("DASD   PAV\n");
  printf("VOLSER IOs Reads      K bytes    Writes     "
         "K bytes    Waits      Average Wait\n");
  printf("------ --- ---------- ---------- ---------- "
         "---------- ---------- ------------\n");

  for (i = 0; i < stdh->number_of_lines; i++, stdd++)
  {
    printf("%6.6s %3u %10u %10u %10u %10u %10u %6u.%3.3u\n",
           stdd->volser,
           stdd->pavios,
           stdd->reads,
           stdd->read_bytes,
           stdd->writes,
           stdd->write_bytes,
           stdd->waits,
           stdd->avg_wait_whole,
           stdd->avg_wait_decimal);
  }
  printf("\n");
  printf("Total number of waits for I/O: %u,,%u\n",
         stdh->grand_total_waits.high, stdh->grand_total_waits.low);
```

```
                    printf("Average I/O wait time:         %9u.%3.3u (msecs)\n",
                           stdh->avg_wait_time_whole,
                           stdh->avg_wait_time_decimal);
                    printf("\n");

                    return 1;
                  }
```

Statistics iocounts Information

Purpose

Displays information about how often zFS performs I/O for various circumstances and how often it waits on that I/O.

Format

```
syscall_parmlist
   opcode               int              243              STATOP_IOCOUNTS
   parms[0]             int              Offset to STAT_API
   parms[1]             int              Offset of output following STAT_API
   parms[2]             int              Offset to system name (optional)
   parms[3]             int              0
   parms[4]             int              0
   parms[5]             int              0
   parms[6]             int              0
STAT_API
   sa_eye               char[4]          "STAP"
   sa_len               int              Length of buffer following STAT_API
   sa_ver               int              1 or 2
   sa_flags             char[1]          0x80 - Reset statistics
   sa_fill              char[3]          Reserved
   sa_supported_ver     int              Version of data returned
   sa_reserve           int[3]           Reserved
   posix_time_high      unsigned int     High order 32 bits since epoch
   posix_time_low       unsigned int     Low order 32 bits since epoch
   posix_useconds       unsigned int     Microseconds
API_IO_BY_TYPE[3]
   number_of_lines      unsigned int     Count of API_IO_BY_TYPE lines (3)
   count                unsigned int     Count of I/Os for type
   waits                unsigned int     Number of waits for type
   cancels              unsigned int     Number of cancels for type
   merges               unsigned int     Number of merges for type
   type                 typechar[6]      Reserved
   description          char[54]         Type description
API_IO_BY_CIRC[19]
   number_of_lines      unsigned int     Count of API_IO_BY_CIRC lines (19)
   count                unsigned int     count of I/Os for circumstance
   waits                unsigned int     Number of waits for circumstance
   cancels              unsigned int     Number of cancels for circumstance
   merges               unsigned int     Number of merges for circumstance
   type                 typechar[6]      Reserved
   description          char[54]         Circumstance description
```

```
-- or --
API_IO_HDR
   number_of_type_lines  unsigned int     Number of API_IO_BY_TYPE2 lines (3)
   number_of_circ_lines  unsigned int     Number of API_IO_BY_CIRC2 lines (19)
   reserved[6]           int              Reserved
API_IO_BY_TYPE2[3]
   count                 unsigned long long  Count of I/Os for type
   waits                 unsigned long long  Number of waits for type
   cancels               unsigned long long  Number of cancels for type
   merges                unsigned long long  Number of merges for type
   type                  char[6]             Reserved
   description           char[54]            Type description
   pad1                  char[4]             Pad bytes
API_IO_BY_CIRC2[19]
   count                 unsigned long long  Count of I/Os for circumstance
   waits                 unsigned long long  Number of waits for circumstance
   cancels               unsigned long long  Number of cancels for circumstance
   merges                unsigned long long  Number of merges for circumstance
   type                  char[6]             Reserved
   description           char[54]            Circumstance description
   pad1                  char[4]             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 notes

- Reserved fields and undefined flags must be set to binary zeros.
- When `sa_supported_ver` is 0 or 1, the output consists of API_IO_BY_TYPE and API_IO_BY_CIRC. When `sa_supported_ver` is 2, the output consists of API_IO_HDR, API_IO_BY_TPYE2, and API_IO_BY_CIRC2

Privilege required

None.

Related services

Statistics iobyaggr Information
Statistics iobydasd 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_IOCOUNTS 243 /* Performance API queries */
```

```
#define TOTAL_TYPES     3
#define TOTAL_CIRC      19
#define SA_VER_INIT     0x01

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 {
  unsigned int            posix_time_high; /*high order 32 bits since epoc*/
  unsigned int            posix_time_low;  /*low order 32 bits since epoch*/
  unsigned int            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_2 0x02
  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_supported_ver; /* version of data returned */
  int               sa_reserve[3];    /* Reserved */
  struct reset_time reset_time_info;
} STAT_API;

typedef struct api_iocount_hdr_2 {
  int number_of_type_lines;
  int number_of_circ_lines;
  int reserved[6];
} API_IOCOUNT_HDR_2;

typedef struct API_IO_BY_TYPE_2_t {
  unsigned long long int count;
  unsigned long long int waits;
  unsigned long long int cancels;  /* Successful cancels of IO */
  unsigned long long int merges;   /* Successful cancels of IO */
  char                   type[6];
  char                   description[54]; /*add 3 bytes for padding */
  char                   reserved[4];
} API_IO_BY_TYPE_2;

typedef struct API_IO_BY_CIRC_2_t {
  unsigned long long int count;
  unsigned long long int waits;
  unsigned long long int cancels;
  unsigned long long int merges;
  char                   type[6];
  char                   description[54];  /*add 3 bytes for padding */
  char                   reserved[4];
} API_IO_BY_CIRC_2;

/* Version 1 structures */
typedef struct API_IO_BY_TYPE_t
```

```
{
  unsigned int      number_of_lines;
  unsigned int      count;
  unsigned int      waits;
  unsigned int      cancels; /* Successful cancels of IO */
  unsigned int      merges;  /* Successful merges of IO */
  char              reserved1[6];
  char              description[51];
  char              pad1[3];
} API_IO_BY_TYPE;

typedef struct API_IO_BY_CIRC_t
{
  unsigned int      number_of_lines;
  unsigned int      count;
  unsigned int      waits;
  unsigned int      cancels;
  unsigned int      merges;
  char              reserved1[6];
  char              description[51];
  char              pad1[3];
} API_IO_BY_CIRC;

/*********************************************************************/
/* The following structures are used to represent cfgop queries      */
/* for iocounts                                                      */
/*********************************************************************/
struct parmstruct {
  syscall_parmlist  myparms;
  STAT_API          myapi;
  API_IOCOUNT_HDR_2 myiocounthdr;
  API_IO_BY_TYPE_2  mystatsbytype[TOTAL_TYPES];
  API_IO_BY_CIRC_2  mystatsbycirc[TOTAL_CIRC];
  char              systemname[9];
} myparmstruct;

int print_iocounts_version1(STAT_API* stapptr);
int print_iocounts_version2(STAT_API          *stapptr,
                            API_IOCOUNT_HDR_2 *hdrptr,
                            API_IO_BY_TYPE_2  *stiotptr,
                            API_IO_BY_CIRC_2  *stiocptr);

int main(int argc, char **argv)
{
  int               bpxrv;
  int               bpxrc;
  int               bpxrs;
  int               i;

  STAT_API          *stapptr  = &(myparmstruct.myapi);
  API_IOCOUNT_HDR_2 *hdrptr   = &(myparmstruct.myiocounthdr);
  API_IO_BY_TYPE_2  *stiotptr = &(myparmstruct.mystatsbytype[0]);
  API_IO_BY_CIRC_2  *stiocptr = &(myparmstruct.mystatsbycirc[0]);
  char              buf[33];

  myparmstruct.myparms.opcode   = STATOP_IOCOUNTS;
  myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
  myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) +
                                  sizeof(STAT_API);
  myparmstruct.myparms.parms[2] = 0;

  /* 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)                                                    */
  /* + (sizeof(API_IOCOUNT_HDR_2                                           */
  /* + (TOTAL_TYPES * sizeof(API_IO_BY_TYPE_2))                                */
  /* + (TOTAL_CIRC * sizeof(API_IO_BY_CIRC_2));                                */

  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_2;
  stapptr->sa_len = (int)(sizeof(API_IOCOUNT_HDR_2)) +
    (TOTAL_TYPES * sizeof(API_IO_BY_TYPE_2)) +
    (TOTAL_CIRC  * sizeof(API_IO_BY_CIRC_2));

  /* 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\n",
           bpxrv, bpxrc, bpxrs);
    return bpxrc;
  }
  else
  {
    /* Check the output that version that was returned */
    if (stapptr->sa_supported_ver == SA_VER_INIT)
      print_iocounts_version1(stapptr);
    else
      print_iocounts_version2(stapptr, hdrptr, stiotptr, stiocptr);

    unsigned int ptl = stapptr->reset_time_info.posix_time_low;
    if (0 == ctime_r((time_t * ) & ptl, 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;
}

int print_iocounts_version1(STAT_API* stapptr)
{
  char *p = (char*) stapptr;
  p += sizeof(STAT_API);
  API_IO_BY_TYPE *stiotptr = (API_IO_BY_TYPE*) p;
  p += sizeof(API_IO_BY_TYPE) * TOTAL_TYPES;
  API_IO_BY_CIRC *stiocptr = (API_IO_BY_CIRC*) p;

  int i;
  printf("Displaying Version 1 Output\n");
  if (stiotptr->number_of_lines != TOTAL_TYPES)
```

```
    {
      printf("Unexpected number of IO Types, %d instead of TOTAL_TYPES\n",
             stiotptr->number_of_lines);
      return 1;
    }
    if (stiocptr->number_of_lines != TOTAL_CIRC)
    {
      printf("Unexpected number of IO Circumstances, %d instead of TOTAL_CIRC\n",
             stiocptr->number_of_lines);
      return 2;
    }
    printf("\n                     I/O Summary By Type\n");
    printf("                     -------------------\n");
    printf("\n");
    printf("Count      Waits      Cancels    Merges     Type      \n");
    printf("---------- ---------- ---------- ---------- ----------\n");

    for (i = 0; i < TOTAL_TYPES; i++)
    {
      printf("%10u %10u %10u %10u %s\n",
             stiotptr->count, stiotptr->waits,
             stiotptr->cancels, stiotptr->merges,
             stiotptr->description);
      stiotptr = stiotptr + 1;
    }

    printf("\n");
    printf("                 I/O Summary By Circumstance\n");
    printf("                 ---------------------------\n");
    printf("\n");
    printf("Count      Waits      Cancels    Merges     Circumstance\n");
    printf("---------- ---------- ---------- ---------- ------------\n");
    for (i = 0; i < TOTAL_CIRC; i++)
    {
      printf("%10u %10u %10u %10u %s\n",
             stiocptr->count, stiocptr->waits,
             stiocptr->cancels, stiocptr->merges,
             stiocptr->description);
      stiocptr = stiocptr + 1;
      printf("\n");
    }
    return 0;
  }

  int print_iocounts_version2(STAT_API          *stapptr,
                              API_IOCOUNT_HDR_2 *hdrptr,
                              API_IO_BY_TYPE_2  *stiotptr,
                              API_IO_BY_CIRC_2  *stiocptr)
  {
    int i;
    if (hdrptr->number_of_type_lines != TOTAL_TYPES)
    {
      printf("Unexpected number of IO Types, %d instead of TOTAL_TYPES\n",
             hdrptr->number_of_type_lines);
      return 1;
    }
    if (hdrptr->number_of_circ_lines != TOTAL_CIRC)
    {
      printf("Unexpected number of IO Circumstances, %d instead of TOTAL_CIRC\n",
             hdrptr->number_of_circ_lines);
      return 2;
    }

    printf("\n                                  I/O Summary By Type\n");
    printf("                                  -------------------\n");
    printf("\n");
    printf("Count                Waits                Cancels              "
           "Merges              Type      \n");
    printf("-------------------- -------------------- -------------------- "
           "-------------------- ----------\n");

    for (i = 0; i < TOTAL_TYPES; i++)
    {
      printf("%20llu %20llu %20llu %20llu %s\n",
             stiotptr->count, stiotptr->waits,
             stiotptr->cancels, stiotptr->merges,
             stiotptr->description);
      stiotptr = stiotptr + 1;
    }

    printf("\n");
    printf("                              I/O Summary By Circumstance\n");
    printf("                              ---------------------------\n");
    printf("\n");
    printf("Count                Waits                Cancels              "
           "Merges              Circumstance\n");
    printf("-------------------- -------------------- -------------------- "
           "-------------------- ------------\n");

    for (i = 0; i < TOTAL_CIRC; i++)
    {
      printf("%20llu %20llu %20llu %20llu %s\n",
             stiocptr->count, stiocptr->waits,
             stiocptr->cancels, stiocptr->merges,
             stiocptr->description);
      stiocptr = stiocptr + 1;
      printf("\n");
    }

    return 0;
  }
```

Statistics Kernel Information

Purpose

A performance statistics operation that returns kernel counters, including the number of kernel operations and average time for the operation.

Format

```
syscall_parmlist
   opcode              int             246    STATOP_KNPFS
   parms[0]            int             Offset to STAT_API
   parms[1]            int             offset of output following STAT_API
   parms[2]            int             Offset to system name (optional)
   parms[3]            int             0
   parms[4]            int             0
   parms[5]            int             0
   parms[6]            int             0
STAT_API
   sa_eye              char[4]         "STAP"
   sa_len              int             Length of buffer following STAT_API
   sa_ver              int             1 or 2
   sa_flags            char[1]         0x80 - Reset statistics
   sa_fill             char[3]         Reserved
   sa_supported_ver    int             Version of data returned or 0
   sa_reserve          int[3]          Reserved
   posix_time_high     unsigned int    High order 32 bits since epoch
   posix_time_low      unsigned int    Low order 32 bits since epoch
   posix_useconds      unsigned int    Microseconds
   pad1                int             Reserved
KERNEL_CALL_STATS
   kc_eye              char[8]         Reserved
   kc_version          short           Reserved
   kc_len              short           Reserved
   pad1                int             Reserved
  KERNEL_LINE[40]
     kl_operation_name char[27]        Operation name string
     kl_valid          char            Operation entry is valid (0x01)
     kl_count          unsigned int    Count of operations
     kl_time           two_words       High - integer part of average time
                                       Low - fractional part of average time
     kl_bytes          hyper           Bytes associated with read and write
                                       operations, 0 otherwise
     kl_reserved       int[6]          Reserved
   kc_totalops         unsigned int    Grand total operations
   pad2                int             Reserved
   kc_totaltime        hyper           High=integer part of average
                                       wait time
                                       Low=fractional part of average
                                       wait time
   kc_valid_slots      int             Number of slots in above array that
                                       actually contains data
   kc_reserved         int[10]         Reserved
   pad3                int             Reserved
```

```
-- or --
KERNEL_CALL_STATS2
   kc_eye                       char [8]            "KCSTAT2"
   kc_version                   short               1
   kc_len                       short               Size of KERNEL_CALL_STATS2
   pad1                         int                 Reserved
   kc_kernel_line_count         unsigned int        Number of KERNEL_LINE2s
                                                    for kernel
   kc_client_line_count         unsigned int        Number of KERNEL_LINE2s
                                                    for clients
   kc_totalops                  unsigned long long  Total operations
   kc_totalxcfops               unsigned long long  Total xcf operations
   kc_client_totalops           unsigned long long  Total operations for
                                                    clients
   kc_client_totalxcfops        unsigned long long  Total xcf operations for
                                                    clients
   kc_totaltime_whole           unsigned int        Whole portion of average
                                                    total time
   kc_totaltime_decimal         unsigned int        Decimal portion of average
                                                    total time
   kc_client_totaltime_whole    unsigned int        Whole portion of average
                                                    client total time
   kc_client_totaltime_decimal unsigned int         Decimal portion of average
                                                    client total time
   kc_reserved[10]              int                 Reserved
KERNEL_LINE2[n]
   kl_operation_name            char[27]            operation name string
   kl_valid                     char                1 - operation entry valid
   pad1                         int                 Reserved
   kl_count                     unsigned long long  Count of operations
   kl_xcfcount                  unsigned long long  Count of xcf operations
   kl_time                      hyper               High=integer part of
                                                    average time
                                                    Low=fractional part of
                                                    average time
   kl_bytes                     unsigned long long  Bytes in read and write
                                                    operations, otherwise 0
   kl_reserved                  int[4]              Reserved
systemname                      char[9]             System to get stats from

Return_value     0 if request is successful, -1 if it is not successful

Return_code
  EINTR          zFS is shutting down
  EINVAL         Invalid parameter list
  EMVSERR        Internal error occurred
  E2BIG          Information too big for buffer supplied
Reason_code
  0xEFnnxxxx     See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.
- When a_supported_ver is 0 or 1, output consists of KERNEL_CALL_STATS and KERNEL_LINE. When sa_supported_ver is 2, output consists of KERNEL_CALL_STATS2 and KERNEL_LINE2.
- When a_supported_ver is 2, the KERNEL_LINE2 follows the KERNEL_CALL_STATS2 structure. There are kc_kernel_line_count KERNEL_LINE2 structures to represent kernel lines of output. These are followed by kc_client_line_count KERNEL_LINE2 structures of client output lines.

Privilege required

None.

Related services

Statistics Vnode Cache Information
Statistics Metadata Cache Information

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>

#define ZFSCALL_STATS 0x40000007
#define STATOP_KNPFS  246
#define BUFFER_SIZE   1024 * 64
#define SA_VER_INIT   0x01

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_t2 {
  char               kl_operation_name[27];
  char               kl_valid;
  int                pad1;
  unsigned long long kl_count;
  unsigned long long kl_xcfcount;
  two_words          kl_time;
  uint64_t           kl_bytes;
  int                kl_reserved[4];
} KERNEL_LINE2;

typedef struct kernel_call_stats_t2 {
  char               kc_eye[8];  /*eye catcher */
  short              kc_version;
  short              kc_len;
  int                pad1;
  int                kc_kernel_line_count;
  int                kc_client_line_count;
  unsigned long long kc_totalops;        /*Owner grand Total operations*/
  unsigned long long kc_totalxcfops;     /*Owner grand Total xcf operations*/
  unsigned long long kc_client_totalops;    /*Client grand Total operations*/
  unsigned long long kc_client_totalxcfops; /*Client grand Total operations*/
  two_words          kc_totaltime;          /*Owner Grand Total wait time*/
  two_words          kc_client_totaltime;   /*Client Grand Total wait time*/
  int                kc_reserved[10];
} KERNEL_CALL_STATS2;

/* Version 1 Output Structures */
typedef struct KERNEL_line_t {
  char              kl_operation_name[27];
  char              kl_valid;
  unsigned int      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];
  unsigned int   kc_totalops;    /*Grand Total operations */
  int            pad2;
  two_words      kc_totaltime;   /*Grand Total wait time*/
  int            kc_valid_slots; /* Number of slots in the above array*/
                                 /* that actually contain data*/
  int            kc_reserved[10];
  int            pad3;
} KERNEL_CALL_STATS;

/* reset timestamp */
typedef struct reset_time {
  unsigned int            posix_time_high; /*high order 32 bits since epoc*/
  unsigned int            posix_time_low;  /*low order 32 bits since epoch*/
  unsigned int            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_2 0x02
  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_supported_ver; /* version of data returned */
  int               sa_reserve[3];    /* Reserved */
  struct reset_time reset_time_info;
} STAT_API;

struct parmstruct {
  syscall_parmlist   myparms;
  STAT_API           myapi;
  KERNEL_CALL_STATS2 mystats;
  KERNEL_LINE2       mykernline;
  char               systemname[9];
} myparmstruct;

int print_stat_kern_version1(STAT_API* stapptr);

int main(int argc, char **argv)
{
  int                bpxrv;
  int                bpxrc;
  int                bpxrs;
  int                i,j;
  int                processing_server_data = 1;
  int                lines;
  int                buff_fill_len;
  char               itoaBuff[11];
  two_words          totaltime;
  unsigned long long totalops;
  unsigned long long totalxcfops;

  STAT_API            local_req;
  char*               buffp   = NULL;
  syscall_parmlist*   parmp   = NULL;
  STAT_API*           stapptr = NULL;
  KERNEL_CALL_STATS2* kcp     = NULL;
  KERNEL_LINE2*       klp     = NULL;
  char                buf[33];

  stapptr = &local_req;
  memset(stapptr, 0x00, sizeof(STAT_API));
  memcpy(stapptr->sa_eye, SA_EYE, 4);

  stapptr->sa_ver = SA_VER_2;
  stapptr->sa_len = ((2 * MAX_KERNEL_LINES) * sizeof(KERNEL_LINE2)) +
    sizeof(KERNEL_CALL_STATS2);

  buffp = (char*) malloc(BUFFER_SIZE);
  if( buffp == NULL )
  {
    printf("Malloc Error\n");
    return 0;
  }
  memset( buffp, 0x00, sizeof(syscall_parmlist) + sizeof(STAT_API));


  parmp = (syscall_parmlist*) &buffp[0];
  parmp->opcode   = STATOP_KNPFS;
  parmp->parms[0] = sizeof(syscall_parmlist);
  parmp->parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
  parmp->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                       */
  /* parmp->parms[2] = sizeof(syscall_parmlist) + sizeof(STAT_API) +      */
  /* sizeof(KERNEL_CALL_STATS2);                                          */

  parmp->parms[3] = 0;
  parmp->parms[4] = 0;
  parmp->parms[5] = 0;
  parmp->parms[6] = 0;

  buff_fill_len = sizeof(syscall_parmlist);
  stapptr = (STAT_API*) &buffp[buff_fill_len];
  memcpy( stapptr, &local_req, sizeof(STAT_API) );
  buff_fill_len += sizeof(STAT_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 */
        BUFFER_SIZE,                /* Length of Argument */
        (char *) buffp,             /* Pointer to Argument */
        &bpxrv,                     /* Pointer to Return_value */
        &bpxrc,                     /* Pointer to Return_code */
        &bpxrs);                    /* Pointer to Reason_code */

if (bpxrv < 0)
{
  printf("Error querying kernel calls, "
         "BPXRV = %d BPXRC = %d BPXRS = %x\n",
         bpxrv, bpxrc, bpxrs);
  return bpxrc;
}
else
{
  if (stapptr->sa_supported_ver == SA_VER_INIT)
  {
    print_stat_kern_version1(stapptr);
  }
  else
  {
    /* Get the pointers to the output structures */
    kcp = (KERNEL_CALL_STATS2*) &buffp[buff_fill_len];
    buff_fill_len += sizeof(KERNEL_CALL_STATS2);
    klp = (KERNEL_LINE2*) &buffp[buff_fill_len];

    lines = kcp->kc_kernel_line_count;
    totaltime = kcp->kc_totaltime;
    totalops = kcp->kc_totalops;
    totalxcfops = kcp->kc_totalxcfops;

    printf("          zFS Kernel PFS Calls\n");
    printf("          ---------------------\n\n");

    /* Do once if no client information,              */
    /* otherwise loop again printing out client stats */
    int do_client = 1;
    while( do_client )
    {
      if( processing_server_data )
        printf("%15c On Owner \n", ' ');
      else
        printf("%15c On Client \n", ' ');

      printf("          ---------------------\n\n");

      printf("Operation          Count       XCF req          "
             "Avg Time    Bytes     \n");
      printf("---------          ----------  ----------       "
             "---------   ----------\n");

      for (j = 0; j < lines; j++)
      {
        if ( !(klp->kl_valid) )
          break;

        sprintf( itoaBuff, "%d", klp->kl_bytes );

        printf("%13s    %10llu   %10llu   %9u.%3.3u   %10s\n",
               klp->kl_operation_name,
               klp->kl_count,
               klp->kl_xcfcount,
                 klp->kl_time.double_word.first_word,
                 klp->kl_time.double_word.second_word,
                 klp->kl_bytes ? itoaBuff : "");
          klp++;
        }

        /* Print out the Totals */
        printf("------------      ----------   ----------      ----------\n");
        printf("%13s     %10llu   %10llu   %9u.%3.3u\n\n\n",
               "TOTALS*",
               totalops,
               totalxcfops,
               totaltime.double_word.first_word,
               totaltime.double_word.second_word);

        /* If client data exists, and we have not already processed it */
        if ( (processing_server_data) && (kcp->kc_client_line_count) )
        {
          /* setup the client data */
          lines = kcp->kc_client_line_count;
          totaltime = kcp->kc_client_totaltime;
          totalops = kcp->kc_client_totalops;
          totalxcfops = kcp->kc_client_totalxcfops;
          processing_server_data = 0;
          do_client = 1;
        }
        else
          do_client = 0;
      }
    }

    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;
}

int print_stat_kern_version1(STAT_API* stapptr)
{
  int i;
  char *p = (char*) stapptr;
  p += sizeof(STAT_API);
  KERNEL_CALL_STATS *stkcptr = (KERNEL_CALL_STATS*) p;

  printf("Displaying the Version 1 Stats\n");
  printf("\n%34s\n", "zFS Kernel PFS Calls");
  printf("%34s\n",   "--------------------");
  printf("\n");
  printf("Operation         Count            Avg Time  \n");
  printf("---------         ----------       ----------\n");

  i = 0;
  while (stkcptr->OUTPUT[i].kl_valid == 1)
  {
    printf("%13s     %10u     %9u.%3.3u\n",
           stkcptr->OUTPUT[i].kl_operation_name,
           stkcptr->OUTPUT[i].kl_count,
           stkcptr->OUTPUT[i].kl_time.double_word.first_word,
           stkcptr->OUTPUT[i].kl_time.double_word.second_word);
    i += 1;
  }
  printf("---------          ----------         ----------\n");
  printf("*TOTALS*           %10u     %9u.%3.3u\n",
         stkcptr->kc_totalops,
         stkcptr->kc_totaltime.double_word.first_word,
         stkcptr->kc_totaltime.double_word.second_word);
}
```

Statistics Locking Information

Purpose

A performance statistics operation that returns locking information. Requesting version 1 output returns counters with 4-byte values. Requesting version 2 output returns counters with 8-byte values.

Format

```
syscall_parmlist
   opcode                          int             240   STATOP_LOCKING
   parm[0]                         int             Offset to STAT_API
   parm[1]                         int             Offset of output following STAT_API
   parm[2]                         int             Offset to system name
   parm[3]                         int             0
   parm[4]                         int             0
   parm[5]                         int             0
   parm[6]                         int             0

STAT_API
   sa_eye                          char[4]         "STAP"
   sa_len                          int             Length of buffer that
                                                   follows STAT_API
   sa_ver                          int             1 or 2
   sa_flags                        char            0x80 for reset; 0 otherwise
   sa_fill                         char[3]         0
   sa_supported_ver                int             Version of data returned (0
                                                   and 1 both mean version 1)
   sa_reserve                      int[3]          0
   posix_time_high                 unsigned int    High order 32 bits since
                                                   epoch
   posix_time_low                  unsigned int    Low order 32 bits since
                                                   epoch
   posix_useconds                  unsigned int    Microseconds
   pad1                            int             Reserved
STAT_LOCKING
   reserved1                       int             Reserved
   stlk_untimed_sleeps             unsigned int    Number of untimed sleeps
   stlk_timed_sleeps               unsigned int    Number of timed sleeps
   stlk_wakeups                    unsigned int    Number of wake ups
   stlk_total_wait_for_locks       unsigned int    total waits for locks
       pad1                       int              Reserved
   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)
   stlk_avg_lock_wait_time_decimal int             Average lock wait time in
                                                   msecs (decimal part in
                                                   thousandths (3 means .003,
                                                   300 means .3)
   stlk_total_monitored_sleeps        unsigned int   Total monitored sleeps
      pad2                           int            Reserved
   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 decimal)
   stlk_avg_mon_sleep_time_decimal    int            Average monitored sleep time
                                                     in msecs. Decimal part is in
                                                     thousandths (3 means .003,
                                                     00 means .3)
   stlk_total_contentions             unsigned int   Total lock contention
   stlk_reserved_space                char[48]       Reserved for future use
        pad3                           int           Reserved
  LOCK_LINE[15]                      struct Lock_line[15]   Lock data
       count                          int            Number of waits for lock
       async                          int            Asynchronous disposition
       spins                          int            Number of attempts to get
                                                     lock that did not resolve
                                                     immediately
       pad                            int            Keep alignment boundaries
       percentage                     double
       percentage_whole               int            Percentage >= 1
       percentage_decimal             int            Percentage < 1. Decimal part
                                                     is in thousandths (3 means
                                                     .003 and 300 means .3)
       description                    char[84]        Description of the lock
       pad2                           int             Reserved

 SLEEP_LINE[5]                       struct Sleep_line[5]  Storage for sleep data
       sleepcount                     unsigned int    Time spent sleeping
       pad                            int             Keep alignment boundaries
       percentage                     double          Percentage of time spent
                                                      sleeping
       percentage_whole               int             Percentage >=1
       percentage_decimal             int             Percentage < 1. Decimal part
                                                      is in thousandths (3 means
                                                      .003 and 300 means .3)
       description                    char[84]         Description of the thread
       pad                            int             Keep alignment boundaries
  systemname char[9]
 --  or  --
STAT_LOCKING2
      reserved1 int[2]                 int
      stlk_untimed_sleeps              unsigned long long int  Untimed sleeps
      stlk_timed_sleeps                unsigned long long int  Timed sleeps
      stlk_wakeups                     unsigned long long int  Wake ups
      stlk_total_wait_for_locks        unsigned long long int  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 part
                                                     is in thousandths (3
                                                     means .003, 300 is .3)
       stlk_total_monitored_sleeps      unsigned long long int  Total monitored
                                                                sleeps
       stlk_average_monitored_sleep_time double      Average monitored sleep time
       stlk_avg_mon_sleep_time_whole    int          Average monitoredsleep time
                                                     in msecs left of the decimal
      stlk_avg_mon_sleep_time_decimal  int          Average monitored
                                                    sleep time in msecs.
                                                    decimal part is in
                                                    thousandths (3 means
                                                    .003, 300 means .3)

      stlk_total_contentions   unsigned long long int   Total lock contention
      stlk_reserved_space      char[48]             Reserved for future
      stlk_lock_line_count     int                  Number of lock lines
      stlk_sleep_line_count    int                  Number of sleep lines

LOCK_LINE2[m]
    count               unsigned long long int     Number of thread waits for
                                                    this lock
    async               unsigned long long int     Asynchronous disposition
    spins               unsigned long long int     Number of attempts to get
                                                    lock that did not
                                                    resolve immediately
    percentage          double
    percentage_whole    int                        Percentage >= 1
    percentage_decimal  int                        Percentage < 1. Decimal part
                                                    is in thousandths
                                                    (3 means .003, 300 means .3)
    description         char[84]                   Description of the lock
    pad                 int                        Fill space to align

SLEEP_LINE2[n]
    sleepcount          unsigned long long int     Time spent sleeping
    percentage          double                     Percentage of time spent
                                                    sleeping
    percentage_whole    int                        Percentage >=1
    percentage_decimal  int                        Percentage < 1. decimal part
                                                    is in thousandths
                                                    (3 means .003, 300 means .3)
    description         char[84]                   Description of the thread
    pad                 int                        Keep boundary alignment
```

Usage notes

- When sa_supported_ver is 0 or 1, the output consists of STAT_LOCKING, followed by one or more LOCK_LINE, followed by one for more SLEEP_LINE. When sa_supported_ver is 2, the output consists of STAT_LOCKING2, followed by one or more LOCK_LINE2, followed by one for more SLEEP_LINE2.
- 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

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
```

```
#define ZFSCALL_STATS  0x40000007
#define STATOP_LOCKING 240 /* Performance API queries */
#define BUFFER_SIZE    1024 * 64
#define TOP15          15

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_2
{
  unsigned long long int count; /* Number of thread waits for this lock  */
  unsigned long long int async; /* Asynchronous disposition */
  unsigned long long int spins; /* Number of attempts to get lock */
                                /* that didnt resolve immediately*/
  double                 reserved;
  int                    percentage_whole;  /* percentage >= 1*/
  int                    percentage_decimal;/* percentage < 1*/
  char                   description[84];   /* Description of the lock */
  int                    pad2;
} LOCK_LINE_2;

typedef struct Sleep_line_2
{
  unsigned long long int sleepcount;    /* Time spent sleeping */
  double         reserved;
  int            percentage_whole;      /* Percentage >=1 */
  int            percentage_decimal;    /* Percentage < 1 */
  char           description[84];       /*Description of the thread*/
  int            pad2;
} SLEEP_LINE_2;

/*Version 1 Output Structures */
typedef struct Lock_line_t {
  int     count; /* Number of thread waits for this lock */
  int     async; /* Asynchronous disposition*/
  int     spins; /* Number of attempts to get lock that
                    did not resolve immediately*/
  int     pad1;
  double  percentage;
  int     percentage_whole;   /* percentage >= 1*/
  int     percentage_decimal; /* percentage < 1*/
                              /* in thousandths.*/
                              /* For example, 3 means .003 and 300 means .3 */
  char   description[84]; /* Description of the lock */
  int    pad2;
} LOCK_LINE;

typedef struct Sleep_line_t {
  unsigned int sleepcount;         /* Time spent sleeping */
  int    pad1;
  double percentage;         /* Percentage of time spent sleeping*/
  int    percentage_whole;   /* Percentage >=1 */
  int    percentage_decimal; /* Percentage < 1 */
                             /* in thousandths.*/
                             /* For example, 3 means .003 and 300 means .3 */
  char   description[84];    /* Description of the thread*/
  int    pad2;
} SLEEP_LINE;

typedef struct stat_locking_t {
  int          reserved1;
  unsigned int stlk_untimed_sleeps;         /* Number of untimed sleeps */
  unsigned int stlk_timed_sleeps;           /* Number of timed sleeps */
  unsigned int stlk_wakeups;                /* Number of wake ups */
  unsigned int 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*/
                                            /*left of the decimal part */
  int     stlk_avg_lock_wait_time_decimal;  /*Average lock wait time in msecs*/
                                            /* decimal portion */
                                            /* in thousandths */
                                            /* for example, 3 means*/
                                            /* .003 and 300 means .3 */
  unsigned int 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 left of the        */
                                            /* decimal part                */
  int    stlk_avg_mon_sleep_time_decimal;   /* Average monitored sleep */
                                            /* time in msecs */
                                            /* decimal portion             */
                                            /* in thousandths              */
                                            /* for example, 3 means .003   */
                                            /* and 300 means .3 */
  unsigned int 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;

/* reset timestamp */
typedef struct reset_time {
  unsigned int            posix_time_high; /* high order 32 bits since epoc */
  unsigned int            posix_time_low;  /* low order 32 bits since epoch */
  unsigned int            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_2 0x02
#define             SA_VER_INIT 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_supported_ver; /* version of data returned */
  int               sa_reserve[3];    /* Reserved */
  struct reset_time reset_time_info;
} STAT_API;

typedef struct api_lock_stats_2
{
  int pad1;
  int ls_total_bytes_of_data;  /* Total bytes of data*/
  unsigned long long int ls_untimed_sleeps;       /* Number of untimed sleeps*/
  unsigned long long int ls_timed_sleeps;         /* Number of timed sleeps */
  unsigned long long int ls_wakeups;              /* Number of wake ups    */
  unsigned long long int ls_total_wait_for_locks; /* Total waits for locks */
  double ls_average_lock_wait_time;     /*Average lock wait time */
  int    ls_avg_lock_wait_time_whole;   /*Average lock wait time in msecs left
                                          of the decimal part*/
  int    ls_avg_lock_wait_time_decimal; /*Average lock wait time in
                                          msecs decimal portion */
  unsigned long long int ls_total_monitored_sleeps; /*Total monitored sleeps */
  double ls_average_monitored_sleep_time;/* Average monitored sleep time */
  int    ls_avg_mon_sleep_time_whole;   /*Average monitored sleep time in msecs
                                         left of the decimal part*/
  int    ls_avg_mon_sleep_time_decimal; /*Average monitored sleep time in msecs
                                          decimal portion */
  unsigned long long int ls_total_contentions;  /*Total lock contention
                                                  of all kinds*/
  char   ls_reserved_space[48]; /* reserved for future use */
#define MAX_LOCKS 15  /* Maximum number of locks in this release*/
#define MAX_SLEEPS 5  /* Maximum number of sleeps in this release*/
  int    ls_lock_line_count;    /* count of lock lines, currently 15 */
  int    ls_sleep_line_count;   /* count of sleep lines,currently 5 */
} API_LOCK_STATS_2;

int print_locking_version1(char *buffp,
                           int  buff_fill_len);
int print_locking_version2(char *buffp,
                           int  buff_fill_len);

int main(int argc, char **argv)
{
  int               bpxrv;
  int               bpxrc;
  int               bpxrs;
  int               i;
  int               buff_fill_len;

  STAT_API          local_req;
  char              *buffp   = NULL;
  syscall_parmlist  *parmp   = NULL;
  STAT_API          *stapptr = NULL;


  stapptr = &local_req;
  memset( stapptr, 0x00, sizeof(STAT_API) );
  memcpy( stapptr->sa_eye, SA_EYE, 4 );

  stapptr->sa_ver = SA_VER_2;
  stapptr->sa_len = 2 * sizeof(API_LOCK_STATS_2) +
    (MAX_LOCKS * sizeof(LOCK_LINE_2))   +
    (MAX_SLEEPS * sizeof(SLEEP_LINE_2));

  buffp = (char*) malloc(BUFFER_SIZE);
  if( buffp == NULL )
  {
    printf("Malloc Error\n");
    return 0;
  }
  memset( buffp, 0x00, sizeof(syscall_parmlist) + sizeof(STAT_API));

  parmp = (syscall_parmlist*) &buffp[0];
  parmp->opcode   = STATOP_LOCKING;
  parmp->parms[0] = sizeof(syscall_parmlist);
  parmp->parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
  parmp->parms[2] = 0;
  parmp->parms[3] = 0;
  parmp->parms[4] = 0;
  parmp->parms[5] = 0;
```

```
    parmp->parms[6] = 0;

    buff_fill_len = sizeof(syscall_parmlist);
    stapptr = (STAT_API*) &buffp[buff_fill_len];
    memcpy( stapptr, &local_req, sizeof(STAT_API) );
    buff_fill_len += sizeof(STAT_API);

    BPX1PCT("ZFS     ",
            ZFSCALL_STATS,              /* Perf statistics operation */
            BUFFER_SIZE,            /* Length of Argument */
            buffp,                      /* Pointer to Argument */
            &bpxrv,                     /* Pointer to Return_value */
            &bpxrc,                     /* Pointer to Return_code */
            &bpxrs);                    /* Pointer to Reason_code */

    if (bpxrv < 0)
    {
      printf("Error querying locking stats, BPXRV = %d BPXRC = %d BPXRS = %x\n",
             bpxrv, bpxrc, bpxrs);
      return bpxrc;
    }
    else
    {
      if( stapptr->sa_supported_ver == SA_VER_INIT )
        print_locking_version1(buffp, buff_fill_len);
      else
        print_locking_version2(buffp, buff_fill_len);
    }
    return 0;
  }

  int print_locking_version2(char *buffp,
                             int  buff_fill_len)
  {
    int i;
    API_LOCK_STATS_2  *stlkptr = NULL;
    LOCK_LINE_2       *llp     = NULL;
    SLEEP_LINE_2      *slp     = NULL;

    /* Point at output structures located in the buffer */
    stlkptr = (API_LOCK_STATS_2*) &buffp[buff_fill_len];
    buff_fill_len += sizeof(API_LOCK_STATS_2);
    llp     = (LOCK_LINE_2*) &buffp[buff_fill_len];
    buff_fill_len += sizeof(LOCK_LINE_2);

    /* Print out the locking statistics */
    printf("%55s\n","Locking Statistics\n\n");
    printf("Untimed sleeps: %20llu   Timed Sleeps: "
           "%20llu   Wakeups: %20llu\n\n",
           stlkptr->ls_untimed_sleeps,
           stlkptr->ls_timed_sleeps,
           stlkptr->ls_wakeups);

    printf("%-42s %20llu\n",
           "Total waits for locks:",
           stlkptr->ls_total_wait_for_locks);

    printf("%-42s %10u.%3.3u (msecs)\n\n",
           "Average lock wait time:",
           stlkptr->ls_avg_lock_wait_time_whole,
           stlkptr->ls_avg_lock_wait_time_decimal);

    printf("%-42s %10llu\n",
           "Total monitored sleeps:",
           stlkptr->ls_total_monitored_sleeps);

    printf("%-42s %10u.%3.3u (msecs)\n\n",
           "Average monitored sleep time:",
           stlkptr->ls_avg_mon_sleep_time_whole,
           stlkptr->ls_avg_mon_sleep_time_decimal);

    printf("%20c       Top %u Most Highly Contended Locks\n", ' ', TOP15);
    printf("  Thread              Async                 "
           "Spin                                         \n");
    printf("   Wait               Disp.                 "
           "Resol.              Pct.     Description   \n");
    printf("--------------------  --------------------  "
           "--------------------  ------   --------------\n");

    /* Iterate through all the LOCK_LINE_2 structures */
    for (i = 0; i < stlkptr->ls_lock_line_count; i++ )
    {
      printf("%20llu  %20llu  %20llu  %3u.%1.1u%%   %.80s\n",
             llp->count, llp->async, llp->spins,
             llp->percentage_whole, llp->percentage_decimal,
             llp->description);
      llp++;
    }
    printf("\n");

    printf("Total lock contention of all kinds: %10llu\n\n",
           stlkptr->ls_total_contentions);
    printf("                           Top 5 Most Common Thread Sleeps\n");
    printf("Thread Wait               Pct.      Description\n");
    printf("---------------------     -------   -----------\n");

    /* Point where the SLEEP_LINE_2 output structures begin in the buffer */
    slp = (SLEEP_LINE_2*) llp;
    for (i = 0; i < stlkptr->ls_sleep_line_count; i++ )
    {
      printf(" %20llu    %3u.%-3.1u%%   %.80s\n\n",
             slp->sleepcount,
             slp->percentage_whole, slp->percentage_decimal,
             slp->description);
      slp++; /* point at next entry */
    }

    return 1;
  }

  int print_locking_version1(char *buffp,
                             int  buff_fill_len)
  {
    int i;
    printf("Version 1 Output is being displayed\n\n");

    STAT_LOCKING *stlkptr;
    stlkptr = (STAT_LOCKING*) &buffp[buff_fill_len];

    printf("\n%50s\n\n", "Locking Statistics");

    printf("Untimed sleeps:               %10u \n",stlkptr->stlk_untimed_sleeps);
    printf("Timed Sleeps:                 %10u \n",stlkptr->stlk_timed_sleeps);
    printf("Wakeups:                      %10u \n\n",stlkptr->stlk_wakeups);

    printf("Total waits for locks:        %10u\n",
           stlkptr->stlk_total_wait_for_locks);
    printf("Average lock wait time:       %6u.%3.3u (msecs)\n\n",
           stlkptr->stlk_avg_lock_wait_time_whole,
           stlkptr->stlk_avg_lock_wait_time_decimal);

    printf("Total monitored sleeps:       %10u\n",
           stlkptr->stlk_total_monitored_sleeps);
    printf("Average monitored sleep time: %6u.%3.3u (msecs)\n",
           stlkptr->stlk_avg_mon_sleep_time_whole,
           stlkptr->stlk_avg_mon_sleep_time_decimal / 1000);

    printf("\n");
    printf("               Top %u Most Highly Contended Locks\n\n", MAX_LOCKS);
    printf("  Thread     Async       Spin                           \n");
    printf("   Wait      Disp.       Resol.    Pct.   Description   \n");
    printf("---------- ----------  ----------  -----  --------------\n");

    for (i = 0; i < MAX_LOCKS; i++)
    {
      printf("%10u %10u  %10u %3u.%1.1u%%  %.80s\n",
             stlkptr->stlk_locks[i].count,
             stlkptr->stlk_locks[i].async,
             stlkptr->stlk_locks[i].spins,
             stlkptr->stlk_locks[i].percentage_whole,
             stlkptr->stlk_locks[i].percentage_decimal / 100,
             stlkptr->stlk_locks[i].description);
    }

    printf("\n");
    printf("Total lock contention of all kinds: u\n",
           stlkptr->stlk_total_contentions);

    printf("\n");
    printf("                 Top %u Most Common Thread Sleeps\n\n",
           MAX_SLEEPS);

    printf("Thread Wait      Pct.      Description\n");
    printf("-----------      -----     -----------\n");

    for (i = 0; i < MAX_SLEEPS; i++)
    {
      printf(" %10u     %3u.%1.1u%%     %.80s\n",
             stlkptr->stlk_sleeps[i].sleepcount,
             stlkptr->stlk_sleeps[i].percentage_whole,
             stlkptr->stlk_sleeps[i].percentage_decimal / 100,
             stlkptr->stlk_sleeps[i].description);
    }

  }
```

Statistics Log Cache Information

Purpose

A performance statistics operation that returns log cache counters, such as the number of requests, hits, and waits on the log buffer cache.

Beginning in z/OS V2R2, a new log caching facility is used. If version 1 output is requested, only the fields `al_buffers` and `al_writtenPages` are filled in with actual data. All other fields are filled in with zeroes. Statistics for the new log caching facility is returned when version 2 output is requested.

Format

```
syscall_parmlist
   opcode                int            247          STATOP_LOG_CACHE
   parms[0]              int            Offset to STAT_API
   parms[1]              int            offset of output following STAT_API
   parms[2]              int            Offset to system name (optional)
   parms[3]              int            0
   parms[4]              int            0
   parms[5]              int            0
   parms[6]              int            0
STAT_API
   sa_eye                char[4]        "STAP"
   sa_len                int            Length of buffer following STAT_API
   sa_ver                int            1 or 2
   sa_flags              char[1]        0x80 - Reset statistics
   sa_fill               char[3]        Reserved
   sa_supported_ver      int            Version returned in output buffer
   sa_reserve            int[3]         Reserved
   posix_time_high       unsigned int   High order 32 bits since epoch
   posix_time_low        unsigned int   Low order 32 bits since epoch
   posix_useconds        unsigned int   Microseconds
   pad1                  int            Reserved
API_LOG_STATS
   al_eye                char[4]                 "ALOG"
   al_size               short                   Size of output
   al_version            char                    Version (1)
   al_reserved1          char                    Reserved byte
   al_buffers            unsigned long long int  Number of buffers used
   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       unsigned long long 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
API_NL_STATS
   nl_eye                   char[4]                 "NLST"
   nl_sizE                  short                   Size of output structure
   nl_version               char                    2
   nl_future                char                    Reesrved for future use
   nl_logs                  unsigned int            Number of log files
   nl_reclaim_pct           unsigned int            Percentage of logs
                                                    reclaimed at log-full time
   nl_blocks_per_pio        unsigned int            Max number of log file
                                                    blocks to write per log IO
   nl_sched_pct             unsigned int            Inactive buffer schedule
                                                    percentage (of log size)
   nl_cachesize             unsigned int            Number of pages in log
                                                    cache
   nl_fixed                 unsigned int            Non-zero if cache
                                                    permanently fixed in memory
   nl_freeitems             unsigned int            Number of unused pages in
                                                    cache
   nl_ios                   unsigned int            Number of I/Os in-progress
   nl_numblks               unsigned int            Number of dirty metadata
                                                    blocks
   nl_future1               unsigned int            Number of unused pages in
                                                    cache
   nl_tran_started          unsigned long long int  Number of started
                                                    transactions
   nl_act_schedules         unsigned long long int  Number of times active
                                                    records scheduled to disk
   nl_comp_schedules        unsigned long long int  Numner of times complete
                                                    records scheduled to disk
   nl_act_pages             unsigned long long int  Number of active pages
                                                    scheduled to disk
   nl_comp_pages            unsigned long long int  Number of completed pages
                                                    scheduled to disk
   nl_tran_merged           unsigned long long int  Number of merged
                                                    transactions
   nl_act_recswrote         unsigned long long int  Number of active records
                                                    written
   nl_comp_recswrote        unsigned long long int  Number of complete tran
                                                    records written
   nl_comp_transize         unsigned long long int  Number of batched/merged
                                                    transactions written
   nl_tran_active_force     unsigned long long int  Number of times an active
                                                    tran forced
   nl_tran_complete_force   unsigned long long int  Number of times a complete
                                                    tran forced
```

```
nl_recoveries          unsigned long long int  Number of times log file
                                               recovery was run
nl_bufupdates          unsigned long long int  Number of buffer updates
nl_bufnew              unsigned long long int  Number of buffer updates
                                               creating new update record
nl_bufavoid            unsigned long long int  Number of buffer updates
                                               avoided due to prior update
nl_bufovlap            unsigned long long int  Number of buffer updates
                                               that had overlap
nl_killavoid           unsigned long long int  Avoided metadata IOs due to
                                               kill-avoid
nl_schedules           unsigned long long int  Number of times older
                                               buffers scheduled to disks
nl_bufsched            unsigned long long int  Number of actual buffers
                                               schedules and also avg.
                                               quicksort size
nl_endmerges           unsigned long long int  Number of times merged
                                               active records with
                                               previously completed active
                                               trans
nl_endmgcnt            unsigned long long int  Number of records merged
                                               active records with
                                               previously completed active
                                               trans
nl_endnew              unsigned long long int  Number of records merged
                                               that were new to prior
                                               completed tran records
nl_endavoid            unsigned long long int  Number of records merged
                                               that could be skipped
                                               because prior completed
                                               record covered it
nl_endovlap            unsigned long long int  Number of records merged
                                               that had overlap with
                                               previously written trans
nl_nbswrites           unsigned long long int  Number of times we added
                                               NBS blocks to active tran
nl_kills               unsigned long long int  Number of kill calls for
                                               buffers deallocated with
                                               tran
nl_forcecomp           unsigned long long int  Number of times a forced
                                               write of buffer forces
                                               complete tran recods to
                                               log
nl_forceact            unsigned long long int  Number of times a forced
                                               write of buffer forces
                                               active tran recods to log
nl_forces              unsigned long long int  Number of force calls
nl_forcewaits          unsigned long long int  Number of times a force has
                                               to wait for in-progress
                                               log pages
nl_hfact               unsigned long long int  Number of times a
                                               handle-full has to write
                                               active records
nl_hfcomp              unsigned long long int  Number of times a
                                               handle-full has to write
                                               comp records
nl_hf                  unsigned long long int  Number of handle full calls
```

```
   nl_hfsched              unsigned long long int  Number of times a
                                                   handle-full had to schedule
                                                   buffers
   nl_hfsched_blocks       unsigned long long int  Number of times a
                                                   handle-full scheduled
                                                   buffers and hence quicksort
                                                   blocks
   nl_sync                 unsigned long long int  Number of times a log sync
                                                   was requested
   nl_bufwaits             unsigned long long int  Number of times had to wait
                                                   for a buffer
   nl_bufmallocs           unsigned long long int  Number of emergency mallocs
                                                   to avoid deadlock
   nl_act_comp_copies      unsigned long long int  Number of times a write to
                                                   active log had to copy
                                                   completed tran bytes
   nl_future2              unsigned long long int[8] Future use
systemname                 char[9]        System name to get stats from
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.
- The output buffer contains an API_LOG_STATS structure when version 1 information is returned; for example, when sa_supported_ver is 0 or 1. Otherwise, it contains an API_NL_STATS structure when sa_supported_ver is 2.
- As previously noted, when V2R2 returns version 1 data in API_LOG_STATS, only the al_buffers and al_writtenPages fields are set.

Privilege required

None.

Related services

Statistics Vnode Cache Information
Statistics Metadata Cache Information

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_STATS    0x40000007
#define STATOP_LOG_CACHE 247 /* Performance API queries */
#define BUFFER_SIZE      1024 * 64

#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 int     high; /* unsigned int  reserved */
  unsigned int     low;
} hyper;

typedef struct API_NL_STATS_t  {
  char        nl_eye[4];         /* Eye catcher = AMET */
#define       NL_EYE "NLST"
  short       nl_size;           /* Size of output structure */
  char        nl_version;        /* Version of statistics returned */
#define       NL_VER_2 2
  char        nl_future;         /* Future use */
  unsigned int nl_logs;          /* Number of log files */
  unsigned int nl_reclaim_pct;   /* Pct. of log reclaimed at log-full time */
  unsigned int nl_blocks_per_pio; /* Max number of log file blocks to write
                                    per log IO */
  unsigned int nl_sched_pct;  /*Inactive buffer schedule pct. (of log size)*/
  unsigned int nl_cachesize;  /*Number of pages in cache*/
  unsigned int nl_fixed;      /*Non-zero if cache permanently fixed in memory*/
  unsigned int nl_freeitems;  /*Number of unused pages in cache*/
  unsigned int nl_ios;        /*Number of IOs in-progress*/
  unsigned int nl_numblks;    /*Number of dirty meta blocks*/
  unsigned int nl_future1;    /*Number of unused pages in cache*/
  unsigned long long int nl_tran_started;       /* Number of started
                                                   transactions */
  unsigned long long int nl_act_schedules;      /* Number of times active
                                                   records scheduled to disk */
  unsigned long long int nl_comp_schedules;     /* Numner of times complete
                                                   records scheduled to disk */
  unsigned long long int nl_act_pages;          /* Number of active pages
                                                   scheduled to disk */
  unsigned long long int nl_comp_pages;         /* Number of completed pages
                                                   scheduled to disk */
  unsigned long long int nl_tran_merged;        /* Number of merged
                                                   transactions */
  unsigned long long int nl_act_recswrote;      /* Number of active records
                                                   written */
  unsigned long long int nl_comp_recswrote;     /* Number of complete tran
                                                   records written */
  unsigned long long int nl_comp_transize;      /* Number of batched/merged
                                                   transactions written */
  unsigned long long int nl_tran_active_force;  /* Number of times an active
                                                   tran forced */
  unsigned long long int nl_tran_complete_force;/* Number of times a complete
                                                   tran forced */
  unsigned long long int nl_recoveries;         /* Number of times log file
                                                   recovery was run */
  unsigned long long int nl_bufupdates;         /* Number of buffer updates */
  unsigned long long int nl_bufnew;             /* Number of buffer updates
                                                   creating new update record*/
  unsigned long long int nl_bufavoid;           /* Number of buffer updates
                                                   avoided due to prior
                                                   update */
  unsigned long long int nl_bufovlap;           /* Number of buffer updates
                                                   that had overlap */
  unsigned long long int nl_killavoid;          /* Avoided metadata IOs due to
                                                   kill-avoid */
  unsigned long long int nl_schedules;          /* Number of times older
                                                   buffers scheduled to disks*/
  unsigned long long int nl_bufsched;           /* Number of actual buffers
                                                   schedules and also avg.
```

```
                                                     quicksort size */
  unsigned long long int nl_endmerges;          /* Number of times merged
                                                   active records with
                                                   previously completed active
                                                   trans */
  unsigned long long int nl_endmgcnt;           /* Number of records merged
                                                   active records with
                                                   previously completed active
                                                   trans */
  unsigned long long int nl_endnew;             /* Number of records merged
                                                   that were new to prior
                                                   completed tran records */
  unsigned long long int nl_endavoid;           /* Number of records merged
                                                   that could be skipped
                                                   because prior completed
                                                   record covered it */
  unsigned long long int nl_endovlap;           /* Number of records merged
                                                   that had overlap with
                                                   previously written trans */
  unsigned long long int nl_nbswrites;          /* Number of times we added
                                                   NBS blocks to active tran */
  unsigned long long int nl_kills;              /* Number of kill calls for
                                                   buffers deallocated with
                                                   tran */
  unsigned long long int nl_forcecomp;          /* Number of times a forced
                                                   write of buffer forces
                                                   complete tran recods to
                                                   log */
  unsigned long long int nl_forceact;           /* Number of times a forced
                                                   write of buffer forces
                                                   active tran recods to log */
  unsigned long long int nl_forces;             /* Number of force calls */
  unsigned long long int nl_forcewaits;         /* Number of times a force has
                                                   to wait for in-progress
                                                   log pages*/
  unsigned long long int nl_hfact;              /* Number of times a
                                                   handle-full has to write
                                                   active records*/
  unsigned long long int nl_hfcomp;             /* Number of times a
                                                   handle-full has to write
                                                   comp records*/
  unsigned long long int nl_hf;                 /* Number of handle full
                                                   calls */
  unsigned long long int nl_hfsched;            /* Number of times a
                                                   handle-full had to schedule
                                                   buffers */
  unsigned long long int nl_hfsched_blocks;     /* Number of times a
                                                   handle-full scheduled
                                                   buffers and hence quicksort
                                                   blocks */
  unsigned long long int nl_sync;               /* Number of times a log sync
                                                   was requested */
  unsigned long long int nl_bufwaits;           /* Number of times had to wait
                                                   for a buffer */
  unsigned long long int nl_bufmallocs;         /* Number of emergency mallocs
                                                   to avoid deadlock */
  unsigned long long int nl_act_comp_copies;    /* Number of times a write to
                                                   active log had to copy
                                                   completed tran bytes */
  unsigned long long int nl_future2[8];         /* Stats for the future */
} API_NL_STATS;


/* Version 1 Output structure */
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;

/* reset timestamp */
typedef struct reset_time {
  unsigned int      posix_time_high; /* high order 32 bits since epoc */
  unsigned int      posix_time_low;  /* low order 32 bits since epoch */
  unsigned int      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 1 or 2 */
#define             SA_VER_2 0x02
#define             SA_VER_INIT 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_supported_ver; /* version of data returned */
  int               sa_reserve[3];    /* Reserved */
  struct reset_time reset_time_info;
} STAT_API;

int print_logcache_version1(char *buffp, int buff_fill_len);
int print_logcache_version2(char *buffp, int buff_fill_len);

int main(int argc, char **argv)
{
  int      bpxrv;
  int      bpxrc;
  int      bpxrs;
  int      i;
  double   temp_ratio;
  int      buff_fill_len;
  int      whole, decimal;
  char     buf[33];

  unsigned long long int temp_hits, temp_total;

  STAT_API          local_req;
  char*             buffp   = NULL;
  syscall_parmlist* parmp   = NULL;
  STAT_API*         stapptr = NULL;
  API_NL_STATS*     nlp     = NULL;
  stapptr = &local_req;
  memset( stapptr, 0x00, sizeof(STAT_API) );
  memcpy( stapptr->sa_eye, SA_EYE, 4 );

  stapptr->sa_ver = NL_VER_2;
  stapptr->sa_len = sizeof(API_NL_STATS);

  buffp = (char*) malloc(BUFFER_SIZE);
  if( buffp == NULL )
  {
    printf("Malloc Error\n");
    return 0;
  }
  memset( buffp, 0x00, sizeof(syscall_parmlist) + sizeof(STAT_API));

  parmp = (syscall_parmlist*) &buffp[0];
  parmp->opcode   = STATOP_LOG_CACHE;
  parmp->parms[0] = sizeof(syscall_parmlist);
  parmp->parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
  parmp->parms[2] = 0;
  parmp->parms[3] = 0;
  parmp->parms[4] = 0;
  parmp->parms[5] = 0;
  parmp->parms[6] = 0;

  buff_fill_len = sizeof(syscall_parmlist);
  stapptr = (STAT_API*) &buffp[buff_fill_len];
  memcpy( stapptr, &local_req, sizeof(STAT_API) );
  buff_fill_len += sizeof(STAT_API);

  BPX1PCT("ZFS     ",
          ZFSCALL_STATS, /* Perf statistics operation */
          BUFFER_SIZE,   /* Length of Argument */
          buffp,         /* 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
  {
    if( stapptr->sa_supported_ver == SA_VER_INIT )
      print_logcache_version1(buffp, buff_fill_len);
    else
      print_logcache_version2(buffp, buff_fill_len);

    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;
}
```

```
int print_logcache_version2(char *buffp, int buff_fill_len)
{
  int      i;
  int      whole, decimal;
  double   temp_ratio;
  unsigned long long int temp_hits, temp_total;
  API_NL_STATS *nlp = NULL;

  /* Set nlp pointer to the output structure in the buffer */
  nlp = (API_NL_STATS*) &buffp[buff_fill_len];
  printf( "%52s\n","Log File Caching Statistics\n" );
  printf( "Logs\n" );
  printf( "--------\n" );
  printf( "%20u : Log files cached \n", nlp->nl_logs );
  printf( "%20llu : Log files recoveries performed \n",nlp->nl_recoveries );
  printf( "%20llu : Log file syncs (filesys quiesce)\n\n", nlp->nl_sync );
  printf( "Policies\n" );
  printf( "--------\n" );

  printf( "%20u : Reclaim pct. (amount reclaimed at log-full time)\n",
          nlp->nl_reclaim_pct );
  printf( "%20u : Maximuem log pages per IO\n",
          nlp->nl_blocks_per_pio );
  printf( "%20u : Inactive buffer schedule pct. (of log size)\n\n",
          nlp->nl_sched_pct );

  printf( "Storage\n" );
  printf( "--------\n" );
  printf( "%20u : Log Cache Size (in 4K pages, fixed=%s)\n",
          nlp->nl_cachesize, nlp->nl_fixed ? "YES" : "NO" );

  temp_hits = nlp->nl_freeitems;
  temp_total = nlp->nl_cachesize;
  if( temp_hits > temp_total )
    temp_hits = temp_total;
  temp_ratio = ((double)temp_hits) / temp_total;
  temp_ratio *= 100.0;

  /* Convert the ratio to ints representing the whole and decimal parts */
  CONVERT_RATIO_TO_INTS(temp_ratio,whole, decimal);
  whole = 100 - whole;

  printf( "%20u : Pct. of cache in-use\n", whole );
  printf( "%20llu : Free page obtain waits\n", nlp->nl_bufwaits );
  printf( "%20llu : Allocations to avoid deadlock\n\n",nlp->nl_bufmallocs );

  printf( "Transactions\n" );
  printf( "------------\n" );
  printf( "%20llu : Transactions started\n", nlp->nl_tran_started );
  printf( "%20llu : Transactions merged\n", nlp->nl_tran_merged );

  temp_total = nlp->nl_comp_schedules;
  temp_hits  = nlp->nl_comp_transize;
  temp_ratio = (temp_total == 0) ? 0.0 : ((double)temp_hits) / temp_total;
  CONVERT_RATIO_TO_INTS(temp_ratio,whole, decimal);
  decimal = decimal / 100;

  printf( "%18u.%1.1u : Average number of transactions batched together\n",
          whole, decimal );
  printf( "%20llu : Sync calls to an active transaction\n",
          nlp->nl_tran_active_force );
  printf( "%20llu : Sync calls to a completed transaction\n\n",
          nlp->nl_tran_complete_force );

  printf( "IOs and Blocks\n");
  printf( "--------------\n");
  printf( "%20u : Log IOs in progress \n",        nlp->nl_ios );
  printf( "%20u : Dirty metadata blocks\n",       nlp->nl_numblks );
  printf( "%20llu : Metadata block kill calls\n", nlp->nl_kills );
  printf( "%20llu : Log File writes initiated\n", nlp->nl_comp_schedules );

  temp_total = nlp->nl_comp_schedules;
  temp_hits  = nlp->nl_comp_pages;
  temp_ratio = (temp_total == 0) ? 0.0 : ((double)temp_hits) / temp_total;
  CONVERT_RATIO_TO_INTS(temp_ratio,whole, decimal);
  decimal = decimal / 100; /* Just want tenths */

  printf( "     %13u.%1.1u : Average number of pages per log write\n",
          whole, decimal );
  printf( "%20llu : Avoided IOs for metadata block due to deallocation\n",
          nlp->nl_killavoid );
  printf( "%20llu : Scheduled not-recently-updated (NRU) metadata blocks\n",
          nlp->nl_schedules );

  temp_total = nlp->nl_schedules;
  temp_hits = nlp->nl_bufsched;
  temp_ratio = (temp_total == 0) ? 0.0 : ((double)temp_hits) / temp_total;
  CONVERT_RATIO_TO_INTS(temp_ratio,whole, decimal);
  decimal = decimal / 100; /* Just want tenths */

  printf( "     %13u.%1.1u : Avergage number of blocks per NRU IO\n",
          whole, decimal );
  printf( "%20llu : Metadata buffers forced to disk\n",
          nlp->nl_forces );

  temp_total = nlp->nl_forces;
  temp_hits = nlp->nl_forcecomp;
  temp_ratio = (temp_total == 0) ? 0.0 : ((double)temp_hits)/temp_total;
  CONVERT_RATIO_TO_INTS(temp_ratio,whole, decimal);
  decimal = decimal / 100; /* Just want tenths */

  printf( "     %13u.%1.1u : Avg where metadata write forced write of log\n",
          whole, decimal );

  temp_hits = nlp->nl_forcewaits;
  temp_total = nlp->nl_forces;

  if( temp_hits > temp_total )
    temp_hits = temp_total;

  temp_ratio = (temp_total == 0) ? 0.0 : ((double)temp_hits)/temp_total;
  temp_ratio *= 100.0;
  CONVERT_RATIO_TO_INTS(temp_ratio,whole, decimal);

  printf( "%18u.%1.1u : Pct. of metadata buffer forces waited on log IO\n",
          whole, decimal );
  printf( "%20llu : Log-full processing calls\n", nlp->nl_hf );
  temp_total = nlp->nl_hf;
  temp_hits  = nlp->nl_hfsched_blocks;
  temp_ratio = (temp_total == 0) ? 0.0 : ((double)temp_hits)/temp_total;
  CONVERT_RATIO_TO_INTS(temp_ratio,whole, decimal);
  decimal = decimal / 100;   /* Just want tenths */

  printf( "%18u.%1.1u : Avg number of metadata blocks "
          "written per log-full\n\n",
          whole, decimal );

  printf("Update Records\n");
  printf("--------------\n");
  temp_total = nlp->nl_comp_schedules;
  temp_hits = nlp->nl_comp_recswrote;
  temp_ratio = (temp_total == 0) ? 0.0 : ((double)temp_hits)/temp_total;
  CONVERT_RATIO_TO_INTS(temp_ratio,whole, decimal);
  decimal = decimal / 100;   /* Just want tenths */
```

```
  printf( "     %13u.%1.1u : Avg number of update records per log IO.\n",
          whole, decimal );
  printf( "%20llu : Number of NBS records written \n", nlp->nl_nbswrites );
  printf( "%20llu : Number of metadata buffer updates \n",
          nlp->nl_bufupdates );
  printf( "%20llu : Number of updates requiring old-byte copying\n",
          nlp->nl_act_comp_copies );
  printf( "%20llu : Avoided buffer update records due to overlap\n",
          nlp->nl_bufavoid );
  printf( "%20llu : Avoided merge update records due to overlap\n\n",
          nlp->nl_endavoid );
}

int print_logcache_version1(char *buffp, int buff_fill_len)
{
  double   temp_ratio;
  int      whole;
  int      decimal;
  API_LOG_STATS *lgstptr = (API_LOG_STATS*) &buffp[buff_fill_len];

  printf("%52s\n", "Log File Caching Statistics");
  printf(" \n");
  printf("Buffers    (K bytes) Requests   Hits       Ratio  Written \n");
  printf("---------- --------- ---------- ---------- ------ ----------\n");

  temp_ratio = (lgstptr->al_lookups.low == 0) ? 0.0 :
             (((double)lgstptr->al_hits.low) /
              lgstptr->al_lookups.low);
  temp_ratio *= 100.0;
  CONVERT_RATIO_TO_INTS(temp_ratio, whole, decimal);
  decimal = decimal / 100; /* Just want tenths */

  printf("%10u %9u %10u %10u %3u.%1.1u%% %10u\n",
         lgstptr->al_buffers.low,
         lgstptr->al_buffers.low * lgstptr->al_buffsize,
         lgstptr->al_lookups.low, lgstptr->al_hits.low,
         whole, decimal, lgstptr->al_writtenPages.low);

  printf(" \n");
  printf("New buffer: log full waits %10u NBS IO waits %10u\n",
         lgstptr->al_fullWaits.low, lgstptr->al_nbsWaits.low);

  printf(" \n");
}
```

Statistics Metadata Cache Information

Purpose

A performance statistics operation that returns metadata cache counters. It is used to determine the number of requests, hits, and discards from the directory cache.

Format

```
syscall_parmlist
   opcode                 int              248   STATOP_META_CACHE
   parms[0]               int              Offset to STAT_API
   parms[1]               int              Offset of output following STAT_API
   parms[2]               int              Offset to system name (optional)
   parms[3]               int              0
   parms[4]               int              0
   parms[5]               int              0
   parms[6]               int              0
STAT_API
   sa_eye                 char[4]          "STAP"
   sa_len                 int              length of buffer following STAT_API
   sa_ver                 int              1 or 2
   sa_flags               char[1]          0x80 -  Reset statistics
   sa_fill                char[3]          Reserved
   sa_supported_ver       int              Version of data returned
   sa_reserve             int[3]           Reserved
   posix_time_high        unsigned int     High order 32 bits since epoch
   posix_time_low         unsigned int     Low order 32 bits since epoch
   posix_useconds         unsigned int     Microseconds
   pad1                   int              Reserved
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       unsigned long long int  Number of buffers in the cache
     buffsize      int                     Size of each buffer in K bytes
     amc_res1      int                     Reserved
     requests      unsigned long long int  Requests to the cache
     hits          unsigned long long int  Hits in the cache
     updates       unsigned long long int  Updates to buffers in the cache
     reserved      int[10]                 Reserved

  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
      reserved              int[10]         Reserved
   am_reserved3             int             Reserved

--- or ---

API_META_STATS2
   am_eye                   char[4]         "AMET"
   am_size                  short           Size of output
   am_version               char            Version
   am_reserved1             char            Reserved byte
  PRIMARY_STATS2
      buffers       unsigned long long int  Number of buffers in the cache
      buffsize      int                     Size of each buffer in K bytes
      amc_res1      int                     Reserved
      requests      unsigned long long int  Requests to the cache
      hits          unsigned long long int  Hits in the cache
      updates       unsigned long long int  Updates to buffers in the cache
      partialwrites unsigned long long int  Times only part of 8K block written
      reserved      int[8]                  Reserved
   am_reserved3     int                     Reserved

  systemname                char[9]         Name of system to get stats from
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.
- When `sa_supported_ver` is 0 or 1, the output buffer contains an API_META_STATS structure. The BACK_STATS structure contains zeros because there is no longer a metaback cache in V2R2. When `sa_supported_ver` is 2, the output buffer contains an API_META_STATS2 structure.

Privilege required

None.

Related services

Statistics Vnode Cache Information
Statistics Metadata Cache Information

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_STATS     0x40000007
#define STATOP_META_CACHE 248 /* Metadata cache (and back cache) stats */
#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 int      high; /* unsigned int  reserved */
  unsigned int      low;
} hyper;

/*********************************************************************/
/* META cache stats, including backing cache.                        */
/*********************************************************************/
typedef struct PRIMARY_STATS2_t
{
  unsigned long long 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 */
  unsigned long long int requests; /* Requests to the cache */
  unsigned long long int hits;     /* Hits in the cache */
  unsigned long long int updates;  /* Updates to buffers in the cache */
  unsigned long long int partialwrites;  /* Only part of 8K block written to
                                            reduce byte transfer. For version 1
                                            always set partialwrites to 0 */
  int      reserved[8];            /* For future use */ /*@F18508S2*/
} PRIMARY_STATS2;

typedef struct API_META_STATS2_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_STATS2 am_primary;        /* Primary space cache statistics */
  int            am_reserved3[10];  /* Reserved for future use */
} API_META_STATS2;

/* reset timestamp */
typedef struct reset_time {
  unsigned int      posix_time_high; /* high order 32 bits since epoc */
  unsigned int      posix_time_low;  /* low order 32 bits since epoch */
  unsigned int      posix_usecs;     /* microseconds */
  int               pad1;
} RESET_TIME;

/* Version 1 Output Structures */
typedef struct PRIMARY_STATS_t {
  hyper buffers;           /* Number of buffers in cache */
  int   buffsize;          /* Size of each buffer in K bytes */
  int   amc_res1;          /* Reserved for future use, zero in version 1 */
  int   requests_reserved; /* Reserved */
  int   requests;          /* Requests to the cache */
  int   hits_reserved;     /* Reserved */
  int   hits;              /* Hits in the cache */
  int   updates_reserved;  /* Reserved */
  int   updates;           /* Updates to buffers in the cache */
  int   reserved[10];      /* For future use */
} PRIMARY_STATS;
```

```
typedef struct BACK_STATS_t {
  hyper buffers;           /* Number of buffers in cache */
  int   buffsize;          /* Size of each buffer in K bytes */
  int   amc_res1;          /* Reserved for future use, zero in version 1 */
  int   requests_reserved; /* Reserved */
  int   requests;          /* Requests to the cache */
  int   hits_reserved;     /* Reserved */
  int   hits;              /* Hits in the cache */
  int   discards_reserved; /* Reserved */
  int   discards;          /* Discards of data from backing cache */
  int   reserved[10];      /* For future use */
} BACK_STATS;

typedef struct API_META_STATS_t {
  char  am_eye[4];        /* Eye catcher = AMET */
#define   MS_EYE "AMET"
  short am_size;          /* Size of output structure */
  char  am_version;       /* Version of stats */
#define   MS_VER_INITIAL 1  /* First version of log stats */
  char  am_reserved1;     /* Reserved byte, 0 in version 1 */
  PRIMARY_STATS am_primary;       /* Primary space cache statistics */
  BACK_STATS    am_back;          /* Backing cache statistics */
  int           am_reserved3[10]; /* Reserved for future use */
} API_META_STATS;

/*********************************************************************/
/* 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 (1 or 2) */
#define             SA_VER_2 0x02
#define             SA_VER_INIT 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_supported_ver; /* version of data returned */
  int               sa_reserve[3];    /* Reserved */
  struct reset_time reset_time_info;
} STAT_API;

typedef struct parmstruct {
  syscall_parmlist myparms;
  STAT_API         myapi;
  API_META_STATS   mystats;
  char             systemname[9];
} myparmstruct;

int print_metadata_version1(API_META_STATS *metastptr);
int print_metadata_version2(API_META_STATS2 *metastptr);

int main(int argc, char **argv)
{
  int      bpxrv;
  int      bpxrc;
  int      bpxrs;
  int      i;
  double   temp_ratio;
  int      whole;
  int      decimal;
  myparmstruct parmstruct;
```

```
STAT_API *stapptr = &(parmstruct.myapi);
char     buf[33];

parmstruct.myparms.opcode   = STATOP_META_CACHE;
parmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
parmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
parmstruct.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   */
/* parmstruct.myparms.parms[2] = sizeof(syscall_parmlist) +        */
/*                               sizeof(STAT_API) +                */
/*                               sizeof(API_META_STATS);           */

parmstruct.myparms.parms[3] = 0;
parmstruct.myparms.parms[4] = 0;
parmstruct.myparms.parms[5] = 0;
parmstruct.myparms.parms[6] = 0;

memset(stapptr, 0, sizeof(STAT_API));
memcpy(stapptr->sa_eye, SA_EYE, 4);
stapptr->sa_ver = SA_VER_2;
stapptr->sa_len = (int)sizeof(API_META_STATS);

/* This next field should only be set if parms[2] is non-zero */
/* strcpy(myparmstruct.systemname,"DCEIMGVQ"); */

BPX1PCT("ZFS     ",
        ZFSCALL_STATS,              /* Perf statistics operation */
        sizeof(myparmstruct),       /* Length of Argument */
        (char *)&parmstruct,        /* 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;
}
else
{
  if( stapptr->sa_supported_ver == SA_VER_INIT )
  {
    API_META_STATS *metastptr1 = &(parmstruct.mystats);
    print_metadata_version1(metastptr1);
  }
  else
  {
    API_META_STATS2 *metastptr = (API_META_STATS2*)&(parmstruct.mystats);
    print_metadata_version2(metastptr);
  }

  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;
```

```
}

int print_metadata_version2(API_META_STATS2 *metastptr)
{
  double   temp_ratio;
  int      whole;
  int      decimal;

  /* Primary cache */
  printf("\n%60s\n", "Metadata Caching Statistics");
  printf(" \n");
  printf("Buffers              (K bytes)           Requests             ");
  printf("Hits                 Ratio  Updates                         \n");
  printf("-------------------- ------------------ -------------------- ");
  printf("-------------------- ------ --------------------            \n");

  temp_ratio = (metastptr->am_primary.requests == 0) ? 0.0 :
    ((double)metastptr->am_primary.hits)  /
    metastptr->am_primary.requests;
  temp_ratio *= 100.0;
  CONVERT_RATIO_TO_INTS(temp_ratio, whole, decimal);
  decimal = decimal / 100; /* Just want tenths */

  printf("%20llu %19llu %20llu %20llu %3u.%1.1u%% %20llu\n",
         metastptr->am_primary.buffers,
         metastptr->am_primary.buffers *
         metastptr->am_primary.buffsize,
         metastptr->am_primary.requests,
         metastptr->am_primary.hits,
         whole, decimal, metastptr->am_primary.updates);
  printf(" \n");
  return 1;
}

int print_metadata_version1(API_META_STATS *metastptr)
{
  double   temp_ratio;
  int      whole;
  int      decimal;
  printf("Version 1 output is being displayed\n\n");

  /* Primary cache */
  printf("\n%44s\n", "Metadata Caching Statistics");
  printf(" \n");
  printf("Buffers    (K bytes) Requests   Hits       Ratio  Updates   \n");
  printf("---------- --------- ---------- ---------- ------ ----------\n");

  temp_ratio = (metastptr->am_primary.requests == 0) ? 0.0 :
    ((double)metastptr->am_primary.hits)  / metastptr->am_primary.requests;
  temp_ratio *= 100.0;
  CONVERT_RATIO_TO_INTS(temp_ratio, whole, decimal);
  decimal = decimal / 100; /* Just want tenths */

  printf("%10u %9u %10u %10u %3u.%1.1u%% %10u\n",
         metastptr->am_primary.buffers.low,
         metastptr->am_primary.buffers.low * metastptr->am_primary.buffsize,
         metastptr->am_primary.requests, metastptr->am_primary.hits,
         whole, decimal, metastptr->am_primary.updates);
  printf(" \n");

  /* Backing cache */
  printf("%48s\n", "Metadata Backing Caching Statistics");
  printf(" \n");
  printf("Buffers    (K bytes) Requests   Hits       Ratio  Discards  \n");
  printf("---------- --------- ---------- ---------- ------ ----------\n");

  if( metastptr->am_back.requests == 0 )
```

```
          temp_ratio = 0.0;
        else
          temp_ratio = 100 * (((double)metastptr->am_back.hits) /
                              metastptr->am_back.requests);

        CONVERT_RATIO_TO_INTS(temp_ratio, whole, decimal);
        decimal = decimal / 100; /* Just want tenths */

        printf("%10u %9u %10u %10u %3u.%1.1u%% %10u\n",
               metastptr->am_back.buffers.low,
               metastptr->am_back.buffers.low * metastptr->am_back.buffsize,
               metastptr->am_back.requests, metastptr->am_back.hits,
               whole, decimal, metastptr->am_back.discards);
        printf(" \n");
      }
```

Statistics Server Token Management Information

Purpose

Returns the server token manager statistics. These statistics can be used to monitor token-related activity for all file systems that are owned on the local server system. It can also be used to monitor token related activity between this local server system and each individual client system that is accessing the file systems that are owned on the local server system.

Format

```
syscall_parmlist

  opcode           int                  252       STATOP_STKM
  parms[0]         int                  offset to STAT_API
  parms[1]         int                  Offset of output following STAT_API
  parms[2]         int                  0
  parms[3]         int                  0
  parms[4]         int                  0
  parms[5]         int                  0
  parms[6]         int                  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
  sa_supported_ver int                  version of data returned
  sa_reserved      int[3]               0
  posix_time_high  unsigned int         high order 32 bits since epoch
  posix_time_low   unsigned int         low order 32 bits since epoch
  posix_useconds   unsigned int         microseconds
  pad1             int

STKM_API_STATS
  st_eye           char[4]              "STKM"
  st_len           short                size of STKM_API_STATS structure
  st_reserved1     char[2]
  st_maxtokens     unsigned long long   Max num of tokens allowed
  st_allocated     unsigned long long   Number of physically allocated
                                        tokens
  st_inuse         unsigned long long   Number of tokens in use
  st_files         unsigned long long   Number of file structures
                                        allocated
  st_obtains       unsigned long long   Number of tokens obtained
  st_returns       unsigned long long   Number of tokens returned
  st_revokes       unsigned long long   Number of tokens revoked
  st_asyncgrants   unsigned long long   Number of async grants requests
  st_gcs           unsigned long long   Number of token garbage collections
  st_reserved2     char[8]
  st_thrashing     unsigned long long   Number of thrashing files
  st_resolution    unsigned long long   Number of thrash resolutions
  st_reserved3     char[40]
```

```
ss_sysinfo       STKM_SYS_STATS[33]
   ss_eye        char[4]              "STSS"
   ss_len        short                size of STKM_SYS_STATS structure
   ss_reserved1  char[2]
   ss_name       char[8]              Sysname
   ss_token      unsigned long long   Number of tokens the
                                      system currently holds
   ss_obtains    unsigned long long   Number of token obtained
   ss_returns    unsigned long long   Number of token returned
   ss_revokes    unsigned long long   Number of token revokes
   ss_asyncgrant unsigned long long   Number of asynchronously
                                      granted tokens
   ss_reserved2  char[16]

ss_thrashing_objs STKM_THRASHING_FILES[64]
   inode         unsigned int         thrashing file inode
   unique        unsigned int         thrashing file uniqueifer
   name          char[45]             name of thrashing file
   reserved      char[3]

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

Reason_code
  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Users of the API supply as input a buffer that contains a `syscall_parmlist` followed by a STAT_API structure. Output is placed in the buffer after the STAT_API structure.
- The output consists of up to 33 STKM_SYS_STATS and up to 64 STKM_THRASHING_FILES structures.
- Unused elements of the `ss_sysinfo` array have an `ss_name` field that consists of hex zeros.
- Unused elements of the `ss_thrashing_objs` array have an inode field with the value 0.

Privilege required

None.

Related services

Query token_cache_size
Set token_cache_size
Statistics Sysplex Client Operations Information
Statistics Sysplex Owner Operations Information

Restrictions

None.

Example

```
#pragma linkage(BPX1PCT, OS)
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include "stdio.h"

#define ZFSCALL_STATS 0x40000007
#define STATOP_STKM   252
#define BUFFER_SIZE   1024 * 64

typedef struct syscall_parmlist_t {
    int opcode;                 /* Operation code to perform */
    int parms[7];               /* Specific to type of operation, */

} syscall_parmlist;

typedef  struct  reset_time {
    unsigned int   posix_time_high;
    unsigned int   posix_time_low;
    unsigned int   posix_usecs;
    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_INIT 0x01
    char    sa_flags;           /* command 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 stkm_sys_stats_t {
    char               ss_eye[4];        /* eye catcher-"STSS" */
#define                ss_EYE "STSS"
    short              ss_len;
    char               ss_reserved1[2];
    char               ss_name[8];       /* Sysname */
    unsigned long long ss_token;         /* Number of tokens the system */
                                         /* currently holds             */
    unsigned long long ss_obtains;       /* Number of token obtained */
    unsigned long long ss_returns;       /* Number of token returned */
    unsigned long long ss_revokes;       /* Number of token revokes */
    unsigned long long ss_asyncgrant;    /* Number of asynchronously */
                                         /* granted tokens           */
    char               ss_reserved2[16];
} STKM_SYS_STATS;

typedef struct stkm_thrashing_files_t
{
    unsigned int inode;
    unsigned int unique;
    char         name[45];
    char         reserved[3];
} STKM_THRASHING_FILES;

#define MAX_THRASHING_FILES     64
#define SYS_MAX_SYSPLEX_SYSTEMS 32     /* Current max # sysplex images*/
typedef struct stkm_api_stats_t
{
    char                 st_eye[4];    /* eye catcher-"STKM" */
```

```
#define                  ST_EYE "STKM"
    short                st_len;
    char                 st_reserved1[2];
    unsigned long long   st_maxtokens; /* Max num of tokens allowed */
    unsigned long long   st_allocated; /* Num. of physically allocated */
                                       /* tokens                        */
    unsigned long long   st_inuse;     /* Number of tokens in use */
    unsigned long long   st_files;     /* Number of file structures */
                                       /* allocated                 */
    unsigned long long   st_obtains;
    unsigned long long   st_returns;
    unsigned long long   st_revokes;
    unsigned long long   st_asyncgrants;
    unsigned long long   st_gcs;
    char                 st_reserved2[8];
    unsigned long long   st_thrashing;
    unsigned long long   st_resolution;
    char                 st_reserved3[40];

    /* 32 sysplex-members + 1 zlc */
    STKM_SYS_STATS       ss_sysinfo[SYS_MAX_SYSPLEX_SYSTEMS+1];
    STKM_THRASHING_FILES ss_thrashing_objs[MAX_THRASHING_FILES];
} STKM_API_STATS;

int main(int argc, char** argv)
{
    int  buff_fill_len = 0;
    int  bpxrv, bpxrc, bpxrs;
    char sysname[9];
    int  title_done;

    STAT_API              local_req;
    STAT_API              *st_req     = NULL;
    syscall_parmlist      *parmp      = NULL;
    STKM_API_STATS        *st_stats   = NULL;
    STKM_SYS_STATS        *ss_stats   = NULL;
    STKM_THRASHING_FILES  *thrashingp = NULL;
    char                  *buffp      = NULL;

    /* Initialize the local_req to 0s */
    st_req = &local_req;
    memset( st_req, 0x00, sizeof(STAT_API) );

    strcpy( local_req.sa_eye, SA_EYE, sizeof(local_req.sa_eye) );
    local_req.sa_len = sizeof(STKM_API_STATS);
    local_req.sa_ver = SA_VER_INIT;

    /* Allocate Buffer */
    buffp = (char*) malloc(BUFFER_SIZE);
    if( buffp == NULL )
    {
        printf("Malloc Error\n");
        return 0;
    }
    memset( buffp, 0x00, sizeof(syscall_parmlist) + sizeof(STAT_API));

    /* Set the run parms */
    parmp = (syscall_parmlist*) &buffp[0];
    parmp->opcode   = STATOP_STKM;
    parmp->parms[0] = buff_fill_len = sizeof(syscall_parmlist);
    parmp->parms[1] = buff_fill_len + sizeof(STAT_API);
    parmp->parms[2] = 0;
    parmp->parms[3] = 0;
    parmp->parms[4] = 0;
    parmp->parms[5] = 0;
    parmp->parms[6] = 0;
```

```
        st_req = (STAT_API*) &buffp[buff_fill_len];

        memcpy( st_req, &local_req, sizeof(STAT_API) );
        buff_fill_len += sizeof(STAT_API);

        BPX1PCT("ZFS     ",
                ZFSCALL_STATS,              /* Aggregate operation */
                BUFFER_SIZE,                /* Length of Argument */
                (char*) buffp,              /* Pointer to Argument */
                &bpxrv,                     /* Pointer to Return_value */
                &bpxrc,                     /* Pointer to Return_code */
                &bpxrs);                    /* Pointer to Reason_code */

        if( bpxrv )
        {
            /* Bad Return code */
            printf("Error requesting info for stkm stats\n");
            printf("Return Value: %d Return Code: %d Reason Code: %x\n",
                   bpxrv, bpxrc, bpxrs);
        }
        else
        {
            /* Success.  Print the information in a table */
            st_stats = (STKM_API_STATS*) &buffp[buff_fill_len];
            ss_stats = st_stats->ss_sysinfo;
            thrashingp = st_stats->ss_thrashing_objs;

            printf("%20c     Server Token Manager (STKM) Statistics\n", ' ');
            printf("%20c     --------------------------------------\n", ' ');
            printf("Maximum tokens:   %20llu       Allocated tokens:   %20llu\n",
                   st_stats->st_maxtokens, st_stats->st_allocated);
            printf("Tokens In Use:    %20llu       File structures:    %20llu\n",
                   st_stats->st_inuse, st_stats->st_files);
            printf("Token obtains:    %20llu       Token returns:      %20llu\n",
                   st_stats->st_obtains, st_stats->st_returns);
            printf("Token revokes:    %20llu       Async Grants:       %20llu\n",
                   st_stats->st_revokes, st_stats->st_asyncgrants);
            printf("Garbage Collects: %20llu       Thrash Resolutions: %20llu\n",
                   st_stats->st_gcs, st_stats->st_resolution);
            printf("Thrashing Files:  %20llu\n\n", st_stats->st_thrashing);

            printf("%30c            Usage Per System:    \n", ' ');
            printf("System    Tokens              Obtains             ");
            printf("Returns              Revokes              Async Grt\n");
            printf("--------  -------------------- --------------------");
            printf("-------------------- -------------------- ");
            printf("--------------------\n");

            for (int i = 0; i < (SYS_MAX_SYSPLEX_SYSTEMS+1); i++)
            {
                if (ss_stats[i].ss_name[0] == '\0')
                    break;

                memcpy(&sysname, &ss_stats[i].ss_name, 8);
                sysname[8] = '\0';

                printf("%8.8s %20llu %20llu %20llu %20llu %20llu\n",
                       sysname,
                       ss_stats[i].ss_token,
                       ss_stats[i].ss_obtains,
                       ss_stats[i].ss_returns,
                       ss_stats[i].ss_revokes,
                       ss_stats[i].ss_asyncgrant);
            }
            printf("\n");

            title_done = 0;
            for (int j = 0; j < MAX_THRASHING_FILES; j++)
            {
                if (thrashingp[j].inode == 0)
                    break;

                if (title_done == 0)
                {
                    printf("              Thrashing Objects:\n");
                    printf("Inode      Uniquifier   File system \n");
                    printf("---------- ----------   --------------------\n");
                    title_done = 1;
                }
                printf("%20u %20u %s\n", thrashingp[j].inode,
                                         thrashingp[j].unique,
                                         thrashingp[j].name);
            }
            if (title_done)
                printf("\n");
        }
        return 0;
    }
```

Statistics Storage Information

Purpose

A performance statistics operation that returns storage information.

STATOP_STORAGE (241) returns below the 2 G bar information.
STATOP_STORAGE (255) returns above the 2 G bar information.

Format

```
syscall_parmlist
   opcode                        int          241  STATOP_STORAGE or
                                              255  STATOP_STORAGE_ABOVE
   parm[0]                       int          Offset to STAT_API
   parm[1]                       int          Offset of output following STAT_API
                                              following STAT_API
   parm[2]                       int          Offset to system name
                                              (optional)
   parm[3]                       int          0
   parm[4]                       int          0
   parm[5]                       int          0
   parm[6]                       int          0
STAT_API
   sa_eye                        char[4]      "STAP"
   sa_len                        int          Length of buffer that follows
                                              the STAT_API
   sa_ver                        int          1 or 2 for STATOP_STORAGE
                                              1 for STATOP_STORAGE_ABOVE
   sa_flags                      char         0x80 for reset; 0x00 otherwise
   sa_fill                       char[3]      Reserved
   sa_supported_ver              int          Version of data returned
   sa_reserve                    int[3]       Reserved
   posix_time_high               unsigned int High order 32 bits since epoch
   posix_time_low                unsigned int Low order 32 bits since epoch
   posix_useconds                unsigned int Microseconds
   pad1                          int          Reserved
API_STOR_STATS
   reserved1                     int
   ss_total_bytes_allocated      unsigned int Total bytes allocated
   ss_total_pieces_allocated     unsigned int Total pieces allocated
   ss_total_allocation_requests  unsigned int Total allocation requests
   ss_total_free_requests        unsigned int Total free requests
   ss_number_of_comp_lines       unsigned int Total number of component
                                              lines in buffer
   ss_reserved_space             char[52]     Reserved for future use
COMP_LINE[n]
   ss_comp_bytes_allocated       int          The number of bytes allocated
                                              by this component
   ss_comp_pieces                int          The number of pieces allocated
   ss_comp_allocations           int          Number of storage allocation
                                              requests done by this component
   ss_comp_frees                 int          The number of storage frees
                                              done by this component
   ss_comp_description           char[84]     The component description
   ss_number_of_detail_lines     int          The number of detail lines
                                              following this component line
```

```
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
-- or --
API_STOR_STATS2
  ss_total_bytes_of_data        unsigned long long int
                                           Total storage allocated.  May
                                           include storage used by other
                                           components in the address space.
  ss_ioefscm_allocated          unsigned long long int
                                           0 for STATOP_STORAGE (241)
                                           Total bytes allocated by IOEFSCM
                                           for STATOP_STORAGE_ABOVE (255)
  ss_ioefscm_heap_allocated     unsigned long long int
                                           Total bytes allocated by the
                                           IOEFSCM heap.
  ss_ioefscm_heap_pieces        unsigned long long int
                                           Total storage pieces in the
                                           IOEFSCM heap.
  ss_ioefscm_heap_allocations   unsigned long long int
                                           Total allocation requests to
                                           IOEFSCM heap.
  ss_ioefscm_heap_frees         unsigned long long int
                                           Total free requests to IOEFSCM
                                           heap.
  ss_ioefskn_allocated          unsigned long long int
                                           0 for STATOP_STORAGE (241)
                                           Total bytes discarded for
                                           STATOP_STORAGE_ABOVE (255)
  ss_ioefskn_heap_allocated     unsigned long long int
                                           Total bytes allocated by the
                                           IOEFSKN heap.
  ss_ioefskn_heap_pieces        unsigned long long int
                                           Total storage pieces in the
                                           IOEFSKN heap.
  ss_ioefskn_heap_allocations   unsigned long long int
                                           Total allocation requests to
                                           IOEFSKN heap.
  ss_ioefskn_heap_frees         unsigned long long int
                                           Total free requests to IOEFSKN
                                           heap.
  ss_ioefskn_heap_discarded     unsigned long long int
                                           0 for STATOP_STORAGE (241)
  ss_number_of_comp_lines       unsigned int
                                           Total number of components
                                           lines in buffer
  pad                           int        Reserved
  ss_reserved_space             char[56]   Reserved for future use
COMP_LINE2[n]
  ss_comp_bytes_allocated       unsigned long long int
                                           The number of bytes allocated
                                           by this component
  ss_comp_pieces                unsigned long long int
                                           The number of pieces allocated
  ss_comp_allocations           unsigned long long int
                                           The number of storage
                                           allocations requests done by
                                           this component
  ss_comp_frees                 unsigned long long int
                                           The number of storage frees
                                           done by this component
  ss_comp_description           char[84]
                                           The component description
  ss_number_of_detail_lines     int
                                           The number of detail lines
                                           following this component line

DETAIL_LINE2[m]
  ss_detail_bytes_allocated     unsigned long long int
                                           Number of bytes allocated
  ss_detail_pieces              unsigned long long int
                                           Number of pieces allocated
  ss_detail_allocations         unsigned long long int
                                           Number of allocation requests
  ss_detail_frees               unsigned long long int
                                           Number of free requests
  ss_detail_description         char[84] description
  ss_detail_reserved            char[4]    Reserved

systemname                      char[9]    System name where the query is ran

Return value     0 if request is successful, -1 if it is not successful

Return code
  EINTR          ZFS is shutting down
  EINVAL         Invalid parameter list
  EMVSERR        Internal error occurred
  E2BIG          Information too big for buffer supplied
Reason code
  0xEFxxnnnn     See z/OS Distributed File Service Messages and Codes
```

Usage notes

- You can specify a buffer that you think might be large enough or you can specify a buffer length of zero. If you get a return code E2BIG, the required size for the buffer is contained in the `sa_len` field.
- Reserved fields and undefined flags must be set to binary zeros.
- When `sa_supported_ver` is 0 or 1, output consists of API_STOR_STATS, COMP_LINE and DETAIL_LINE. When `sa_supported_ver` is 2, output consists of API_STOR_STATS2, COMP_LINE2 and DETAIL_LINE2.
- For STATOP_STORAGE_ABOVE, `sa_supported_ver` is 1 and output consists of API_STOR_STATS2, COMP_LINE2 and DETAIL_LINE2.

Privilege required

None.

Related services

Statistics Locking Information
Statistics User Cache Information

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)
```

```
extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_STATS        0x40000007
#define STATOP_STORAGE       241 /* below-bar storage stats */
#define STATOP_STORAGE_ABOVE 255
#define STATOP_LAST          STATOP_STORAGE_ABOVE
#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 {
  unsigned int      posix_time_high; /* high order 32 bits since epoc */
  unsigned int      posix_time_low;  /* low order 32 bits since epoch */
  unsigned int      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_2 0x02
#define             SA_VER_INIT 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_supported_ver; /* version of data returned */
  int               sa_reserve[3];    /* Reserved */
  struct reset_time reset_time_info;
} STAT_API;

typedef struct comp_line_2
{
  unsigned long long int ss_comp_bytes_allocated; /* Number of bytes   */
                                                  /* allocated         */
                                                  /* by this component */
  unsigned long long int ss_comp_pieces; /* The number of pieces allocated */
  unsigned long long int ss_comp_allocations; /* the number of storage     */
                                              /* allocations requests done */
                                              /* by this component         */
  unsigned long long int ss_comp_frees; /* 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_2;

typedef struct detail_line_2
{
  unsigned long long int ss_detail_bytes_allocated; /* number of bytes */
```

```
                                                         /* allocated       */
  unsigned long long int ss_detail_pieces;      /*number of pieces allocated*/
  unsigned long long int ss_detail_allocations; /*number of allocation */
                                                /*requests             */
  unsigned long long int ss_detail_frees;       /*number of free requests*/
  char     ss_detail_description[84];           /*description */
  char     ss_reserved_pad[4];
} DETAIL_LINE_2;

typedef struct api_stor_stats_2
{
  /* Total storage allocated, this comes from OS data structures */
  /* and is via a query from OS and may include storage          */
  /* used by other OS components in the address space            */
  /* QUERY,STORAGE equivalent:                                   */
  /*    Total Storage Above/Below 2G Bar Allocated               */
  unsigned long long int ss_total_bytes_of_data;

  /* Total number of bytes allocated by IOEFSCM                  */
  /* The number of bytes allocated via IARV64 by/for program IOEFSCM */
  /* This field valid only for an above-bar storage query        */
  /* QUERY,STORAGE equivalent:                                   */
  /*    Total Bytes Allocated by IOEFSCM (Stack + Heap)          */
  unsigned long long int ss_ioefscm_allocated;

  /* Total number of bytes allocated by IOEFSCM heap */
  /* The number of bytes allocated via calls to obtain storage for IOEFSCM */
  /* QUERY,STORAGE equivalent:              */
  /*    IOEFSCM Heap Bytes Allocated        */
  unsigned long long int ss_ioefscm_heap_allocated;

  /* Total number of storage pieces in IOEFSCM heap */
  /* The number of pieces of allocated storage from calls to obtain storage */
  /* for IOEFSCM */
  /* QUERY,STORAGE equivalent:              */
  /*    IOEFSCM Heap Pieces Allocated       */
  unsigned long long int ss_ioefscm_heap_pieces;

  /* Total number of allocation requests to IOEFSCM heap since
  /* last stats reset */
  /* QUERY,STORAGE equivalent:              */
  /*    IOEFSCM Heap Allocation Requests    */
  unsigned long long int ss_ioefscm_heap_allocations;

  /* Total number of free requests for IOEFSCM heap since last stats reset */
  /* QUERY,STORAGE equivalent:              */
  /*    IOEFSCM Heap Free Requests      */
  unsigned long long int ss_ioefscm_heap_frees;

  /* Total number of bytes allocated by IOEFSKN                  */
  /* The number of bytes allocated via IARV64 by/for program IOEFSKN */
  /* This field valid only for an above-bar storage query        */
  /* QUERY,STORAGE equivalent:                                   */
  /*    Total Bytes Allocated by IOEFSKN (Stack + Heap)          */
  unsigned long long int ss_ioefskn_allocated;

  /* Total number of bytes allocated by IOEFSKN heap */
  /* The number of bytes allocated via calls to obtain storage for IOEFSKN */
  /* QUERY,STORAGE equivalent:              */
  /*    IOEFSKN Heap Bytes Allocated        */
  unsigned long long int ss_ioefskn_heap_allocated;

  /* Total number of storage pieces in IOEFSKN heap */
  /* The number of pieces of allocated storage from calls to obtain */
  /* storage for IOEFSKN */
  /* QUERY,STORAGE equivalent:              */
  /*    IOEFSKN Heap Pieces Allocated       */
  unsigned long long int ss_ioefskn_heap_pieces;

  /* Total number of allocation requests to IOEFSKN heap since */
  /* last stats reset */
  /* QUERY,STORAGE equivalent:                 */
  /*    IOEFSKN Heap Allocation Requests      */
  unsigned long long int ss_ioefskn_heap_allocations;

  /* Total number of free requests for IOEFSKN heap since last stats reset */
  /* QUERY,STORAGE equivalent:                 */
  /*    IOEFSKN Heap Free Requests      */
  unsigned long long int ss_ioefskn_heap_frees;

  /* Total number of bytes discarded via IARV64 DISCARD function */
  /* ... valid only for above-bar storage query.                 */
  /* QUERY,STORAGE equivalent:                 */
  /*    Total Bytes Discarded (unbacked) by IOEFSKN   */
  unsigned long long int ss_ioefskn_heap_discarded;

  /* Total number of components lines in buffer*/
  unsigned int      ss_number_of_comp_lines;
  int pad;
  char   ss_reserved_space[48];   /* reserved for future use */
  char   ss_returned_data[1];     /* start of buffer to put data into */
  char   ss_reserved_pad[7];      /* sizeof() will return size including */
                                  /* these 7 bytes */
} API_STOR_STATS_2;

/* Version 1 Output Structures */

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 int ss_total_bytes_allocated;  /* Total bytes allocated*/
    unsigned int ss_total_pieces_allocated; /* Total pieces allocated*/
    unsigned int ss_total_allocation_requests; /*Total allocation requests*/
    unsigned int ss_total_free_requests;       /*Total free requests*/
    unsigned int 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_2 + COMP_LINE_2s and DETAIL_LINE_2s */
  char             systemname[9];
} myparmstruct;

int print_storage_version1(struct parmstruct *buffp, int buflen);
int print_storage_version2(struct parmstruct *buffp,int buflen,int above_bar);

int main(int argc, char **argv)
{
  int               buffer_success = 0;
  int               above_bar      = 0;
  int               bpxrv;
  int               bpxrc;
  int               bpxrs;
  int               i,j,t;

  char              buf[33];
  struct parmstruct *myp      = &myparmstruct;
  int               mypsize;
  int               buflen;

  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:                      */
  /* 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_2;
  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 */

  for(t = 0; t < 1000 && buffer_success == 0 && above_bar < 2; t++)
```

```
        {
          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);

              free(myp);
              myp = (struct parmstruct *)malloc((int)mypsize);
              memset(myp, 0, mypsize);
              printf("Need buffer size of %d, for a total of %d\n\n",
                     buflen, mypsize);

              /* Base the opcode on the type of storage needed*/
              if( above_bar == 0 )
                myp->myparms.opcode = STATOP_STORAGE;
              else
                myp->myparms.opcode = STATOP_STORAGE_ABOVE;

              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:         */
              /* 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_len = buflen;

              /* Above bar storage needs SA_VER_INIT*/
              stapptr->sa_ver = above_bar == 0 ? SA_VER_2 : SA_VER_INIT;

              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 */

              if( bpxrv != 0 && bpxrc == E2BIG )
                printf("E2BIG: %d times total\n", t++);
              else if( bpxrv == 0 )
              {
                buffer_success = 1;
                bpxrv = -1;

                /*If version 1, either above bar stats or downlevel system*/
                if( stapptr->sa_supported_ver == SA_VER_INIT)
                  above_bar ? print_storage_version2(myp, buflen, above_bar) :
                              print_storage_version1(myp, buflen);
                else if (stapptr->sa_supported_ver == SA_VER_2 )
                {
                  /* First pass get below the bar */
                  print_storage_version2(myp, buflen, above_bar);
                  buffer_success = 0;
                  above_bar += 1;
                }

                unsigned int ptl = stapptr->reset_time_info.posix_time_low;
                if (0 == ctime_r((time_t * ) & ptl, 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);
            }

            free(myp);
            return bpxrc;
          }
        }
        if( t == 1000 )
          printf("Number of failed buffer resizes exceeded.\n");

        free(myp);
        return 0;
      }

      int print_storage_version2(struct parmstruct *buffp, int buflen, int above_bar)
      {
        int                  i,j;
        API_STOR_STATS_2  *stst;
        COMP_LINE_2       *stcl;
        DETAIL_LINE_2     *stdl;
        char              *stsy;

        stst = (API_STOR_STATS_2*) ((char *) buffp +
               sizeof(syscall_parmlist) + sizeof(STAT_API));
        stsy = (char *) ((char *) buffp +
               sizeof(syscall_parmlist) + sizeof(STAT_API) + buflen);

        if (above_bar)
          printf("                 zFS Primary Address Space >2G Stge Usage\n");
        else
          printf("                 zFS Primary Address Space <2G Stge Usage\n");
        printf("                 ----------------------------------------\n");
        printf(" \n");

        if (above_bar)
          printf("Total Storage Above 2G Bar Allocated:          %12llu\n",
                 stst->ss_total_bytes_of_data);
        else
          printf("Total Storage Below 2G Bar Allocated:          %12llu\n\n",
                 stst->ss_total_bytes_of_data);

        if (above_bar)
          printf("Total Bytes Allocated by IOEFSCM (Stack+Heap): %12llu\n",
                 stst->ss_ioefscm_allocated);

        printf("IOEFSCM Heap Bytes Allocated:                  %12llu\n",
               stst->ss_ioefscm_heap_allocated);
        printf("IOEFSCM Heap Pieces Allocated:     %12llu\n",
               stst->ss_ioefscm_heap_pieces);
        printf("IOEFSCM Heap Allocation Requests   %12llu\n",
               stst->ss_ioefscm_heap_allocations);
        printf("IOEFSCM Heap Free Requests         %12llu\n",
               stst->ss_ioefscm_heap_frees);
        printf("\n");

        if (above_bar)
        {
          printf("Total Bytes Allocated by IOEFSKN (Stack+Heap): %12llu\n",
                 stst->ss_ioefskn_allocated);
          printf("Total Bytes Discarded (unbacked) by IOEFSKN:   %12llu\n",
                 stst->ss_ioefskn_heap_discarded);
        }
        printf("IOEFSKN Heap Bytes Allocated:                  %12llu\n",
               stst->ss_ioefskn_heap_allocated);
        printf("IOEFSKN Heap Pieces Allocated:     %12llu\n",
               stst->ss_ioefskn_heap_pieces);
        printf("IOEFSKN Heap Allocation Requests   %12llu\n",
               stst->ss_ioefskn_heap_allocations);
        printf("IOEFSKN Heap Free Requests         %12llu\n",
               stst->ss_ioefskn_heap_frees);

        /* Point the comp_line to the ss_returned_data value */
        /* instead of adding sizeof(API_STOR_STATS_2)        */
        stcl = (COMP_LINE_2*) stst->ss_returned_data;

        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");
          printf("Allocated  Pieces Allocs Frees  Component\n");
          printf("---------- ------ ------ ------ ---------\n");
          printf("\n");

          printf("%10llu %6llu %6llu %6llu %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_2 * )((char *)stcl + sizeof(COMP_LINE_2));
          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("%10llu %6llu %6llu %6llu %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_2 *) stdl;
        }
        printf("\n");

      }

      int print_storage_version1(struct parmstruct *buffp, int buflen)
      {
        int                i,j;
        COMP_LINE         *stcl;
        DETAIL_LINE       *stdl;
        char              *stsy;
        API_STOR_STATS    *stst;

        printf("Version 1 Output is being displayed\n\n");

        stst = (API_STOR_STATS * )((char *)buffp + sizeof(syscall_parmlist) +
                                   sizeof(STAT_API));
        stsy = (char *)((char *)buffp + sizeof(syscall_parmlist) +
                        sizeof(STAT_API) + buflen);

        printf("%18czFS Primary Address Space Storage Usage\n", ' ');
        printf("%18c---------------------------------------\n", ' ');
        printf("\n");
        printf("Total Bytes Allocated: %u (%uK) (%uM)\n",
               stst->ss_total_bytes_allocated,
               stst->ss_total_bytes_allocated / 1024,
               stst->ss_total_bytes_allocated / (1024 * 1024));
        printf("Total Pieces Allocated: %u\n",
               stst->ss_total_pieces_allocated);
        printf("Total Allocation Requests: %u\n",
               stst->ss_total_allocation_requests);
        printf("Total Free Requests: %u, %u\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");
          printf("Allocated  Pieces Allocs Frees  Component\n");
          printf("---------- ------ ------ ------ ---------\n");
          printf("\n");
          printf("%10u %6u %6u %6u %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("%10u %6u %6u %6u %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");

      }
```

Statistics Sysplex Client Operations Information

Purpose

Returns information about the number of local operations that required the sending of a message to another system.

Format

```
syscall_parmlist
  opcode              int               253         STATOP_CTKC
  parms[0]            int               offset to   STAT_API
  parms[1]            int               Offset of output following STAT_API
  parms[2]            int               0
  parms[3]            int               0
  parms[4]            int               0
  parms[5]            int               0
  parms[6]            int               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_supported_ver    int               version of data returned or reserved
  sa_reserve          int[3]            0
  posix_time_high     unsigned int      high order 32 bits since epoch
  posix_time_low      unsigned int      low order 32 bits since epoch
  posix_useconds      unsigned int      microseconds
  pad1                int

CT_HEADER
  ct_eye              char[4]           "CTHD"
  ct_length           short
  ct_version          short
  number_of_ct_sys    unsigned int
  number_of_ct_call   unsigned int

CT_SYS_STATS[number_of_ct_sys]
  cs_eye              char[4]           "CTSY"
  cs_length           short
  cs_version          short
  cs_sysname          char[9]           Name of system.  A value of 0
                                        means there is no information in
                                        this record and any subsequent
                                        record (end of list)
  reserved            char[7]
```

```
CT_CALL_STATS[number_of_ct_call]
  cc_eye              char[4]              "CTCL"
  cc_length           short                Length of structure
  cc_version          short                Structure version
  cc_count            unsigned long long   Number of calls of that type
                                           since last statistics reset.
  cc_xcfreq           unsigned long long   Indicates if an XCF request
                                           was required to process the call.
                                           Always equal tocc_count.
  cc_qwait            unsigned long long   Number of times a request had
                                           to wait in queue before being
                                           dispatched to a processing
                                           task at the owner. Invalid for
                                           this report, will be equal to 0.
  cc_avg_wait_whole   int                  Average time for system to
                                           process call in milliseconds.
                                           This will be round-trip call time
                                           (which includes XCF transmission
                                           time)  This is the part before
                                           the decimal point.
  cc_avg_wait_decimal int                  The part after the decimal
                                           point for average wait time.
                                           This is microseconds.
  cc_name             char[25]
  reserved            char[7]

Return_value     0 if request is successful, -1 if it is not successful

Return_code
  EINTR          zFS is shutting down
  EINVAL         Invalid parameter list
  EMVSERR        Internal error occurred
  E2BIG          Information too big for buffer supplied
Reason_code
  0xEFnnxxxx     See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Users of the API supply as input a buffer that contains a `syscall_parmlist`, followed by a STAT_API structure, followed by an output buffer.
- The output consists of a CT_HEADER followed by an array of CT_SYS_STATS structures and an array of CT_CALL_STATS structures. The number of elements in each array is returned in `number_of_ct_sys` and `number_of_ct_call` respectively.
- If the output buffer is not large enough to contain all of the output, E2BIG is returned and the required size is placed in `sa_len`. The caller can then try the request again with a larger buffer.
- A CT_SYS_STATS structure is returned only for systems that the local client system sent messages to since the last statistics reset.

Privilege required

None.

Related services

Statistics Sysplex Owner Operations Information
Statistics Server Token Management Information

Restrictions

None.

Example

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_STATS 0x40000007
#define STATOP_CTKC   253   /* outbound calls to remote owners */
#define E2BIG         145

typedef struct syscall_parmlist_t {
  int opcode;                 /* Operation code to perform */
  int parms[7];               /* Specific to type of operation, */

} syscall_parmlist;

typedef  struct  reset_time {
  unsigned int  posix_time_high;
  unsigned int  posix_time_low;
  unsigned int  posix_usecs;
  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_INIT 0x01
  char    sa_flags;           /* command 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 CT_CALL_STATS_t {
  char  cc_eye[4];
#define   CC_EYE "CTCL"
  short cc_length;
  short cc_version;
#define   CC_VER_INITIAL 1
  unsigned long long cc_count;
  unsigned long long cc_xcfreq;
  unsigned long long cc_qwait; /* number of waits */
  int   cc_avg_wait_whole;     /* average wait time for calls */
                               /* of this type */
  int   cc_avg_wait_decimal;
  char  cc_name[25];
  char  reserved[7];
} CT_CALL_STATS;

typedef struct CT_SYS_STATS_t {
  char  cs_eye[4];
#define CS_EYE "CTSY"
  short cs_length;
  short cs_version;
#define CS_VER_INITIAL 1
```

```
  char  cs_sysname[9];
  char  reserved[7];
} CT_SYS_STATS;

typedef struct CT_HEADER_t {
  char        ct_eye[4];
#define       CT_EYE "CTHD"
  short       ct_length;
  short       ct_version;
#define       CT_VER_INITIAL 1
  unsigned int number_of_ct_sys;
  unsigned int number_of_ct_call;
} CT_HEADER;

int main(int argc, char** argv)
{
  int  buff_fill_len = 0;
  int  buffer_success = 0;
  int  bpxrv, bpxrc, bpxrs;
  char sysname[9];
  int  num_systems;
  int  num_calls;
  int  entry_size;
  int  mypsize;
  int  buflen;
  int  i,j,t;

  STAT_API          local_req;
  STAT_API*         st_req     = NULL;
  syscall_parmlist* parmp      = NULL;
  CT_HEADER*        ct_p       = NULL;
  CT_SYS_STATS*     ct_sysp    = NULL;
  CT_CALL_STATS*    ct_callp   = NULL;
  char*             p          = NULL;
  char*             buffp      = NULL;

  /* Initialize the local_req to 0s */
  st_req = &local_req;
  memset( st_req, 0x00, sizeof(STAT_API) );

  strcpy( local_req.sa_eye, SA_EYE, sizeof(local_req.sa_eye) );
  local_req.sa_len = 0;
  local_req.sa_ver = SA_VER_INIT;

  /* Allocate Buffer */
  buffp = (char*) malloc(sizeof(syscall_parmlist) + sizeof(STAT_API));
  if( buffp == NULL )
  {
    printf("Malloc Error\n");
    return 0;
  }
  memset( buffp, 0x00, sizeof(syscall_parmlist) + sizeof(STAT_API));

  /* Set the run parms */
  parmp = (syscall_parmlist*) &buffp[0];
  parmp->opcode   = STATOP_CTKC;
  parmp->parms[0] = buff_fill_len = sizeof(syscall_parmlist);
  parmp->parms[1] = buff_fill_len + sizeof(STAT_API);
  parmp->parms[2] = 0;
  parmp->parms[3] = 0;
  parmp->parms[4] = 0;
  parmp->parms[5] = 0;
  parmp->parms[6] = 0;

  st_req = (STAT_API*) &buffp[buff_fill_len];
  memcpy( st_req, &local_req, sizeof(STAT_API) );
  buff_fill_len += sizeof(STAT_API);
```

```
BPX1PCT("ZFS     ",
        ZFSCALL_STATS,      /* Aggregate operation */
        buff_fill_len,      /* Length of Argument */
        (char*) buffp,      /* Pointer to Argument */
        &bpxrv,             /* Pointer to Return_value */
        &bpxrc,             /* Pointer to Return_code */
        &bpxrs);            /* Pointer to Reason_code */

for(t = 0; t < 1000 && buffer_success == 0; t++)
{
  if( bpxrv < 0 )
  {
    /* Look for E2BIG to get the required file size back in the st_req */
    if( bpxrc == E2BIG )
    {
      buflen  = st_req->sa_len;
      mypsize = sizeof(syscall_parmlist) + sizeof(STAT_API) + buflen;

      free(buffp);

      buffp = (char*) malloc(mypsize);
      if( buffp == NULL )
      {
        printf("Malloc Error\n");
        return 0;
      }
      memset( buffp, 0x00, mypsize );
      printf("Need buffer size of %d, for a total of %d\n",
             buflen, mypsize);

      /* Set the run parms */
      parmp = (syscall_parmlist*) &buffp[0];
      parmp->opcode   = STATOP_CTKC;
      parmp->parms[0] = buff_fill_len = sizeof(syscall_parmlist);
      parmp->parms[1] = buff_fill_len + sizeof(STAT_API);
      parmp->parms[2] = 0;
      parmp->parms[3] = 0;
      parmp->parms[4] = 0;
      parmp->parms[5] = 0;
      parmp->parms[6] = 0;

      st_req = (STAT_API*) &buffp[buff_fill_len];
      memcpy( st_req->sa_eye, SA_EYE, 4 );
      buff_fill_len += sizeof(STAT_API);
      st_req->sa_ver = SA_VER_INIT;
      st_req->sa_len = buflen;

      BPX1PCT("ZFS     ",
              ZFSCALL_STATS,              /* Aggregate operation */
              mypsize,                    /* Length of Argument */
              (char*) buffp,              /* Pointer to Argument */
              &bpxrv,                     /* Pointer to Return_value */
              &bpxrc,                     /* Pointer to Return_code */
              &bpxrs);                    /* Pointer to Reason_code */

      if( bpxrv != 0 && bpxrc == E2BIG )
        printf("E2BIG: %d times total\n", t++);
      else if( bpxrv == 0 )
      {
        buffer_success = 1;
        ct_p = (CT_HEADER*) &buffp[buff_fill_len];
        buff_fill_len += ct_p->ct_length;
        ct_sysp = (CT_SYS_STATS*) &buffp[buff_fill_len];
        buff_fill_len += ct_sysp->cs_length;
        ct_callp = (CT_CALL_STATS*) &buffp[buff_fill_len];
        /* Make sure there are systems */
        num_systems = ct_p->number_of_ct_sys;
        if( num_systems == 0 )
        {
          printf("Ctkc completed successfully. "
                 "There is no information to display\n");
          free(buffp);
          return 0;
        }
        num_calls  = ct_p->number_of_ct_call;
        entry_size = ct_sysp->cs_length +
          (ct_callp->cc_length * num_calls);

        for (j = 0; j < num_systems; j++)
        {
          printf("CS");
          printf("%5c          SVI Calls to System %s\n", ' ',
                 ct_sysp->cs_sysname );
          printf("  ");
          printf("%15c----------------------------\n", ' ');
          printf("SVI Call                   Count"
                 "                  Avg. Time\n");
          printf("--------------------       --------------------"
                 "   ----------\n");

          for (i = 0; i < num_calls-1; i++)
          {
            printf("%-25s  %20llu %8u.%3.3u\n",
                   ct_callp[i].cc_name,
                   ct_callp[i].cc_count,
                   ct_callp[i].cc_avg_wait_whole,
                   ct_callp[i].cc_avg_wait_decimal);
          }

          /* Put out the Totals entry */
          printf("--------------------       --------------------"
                 "   ----------\n");
          printf("%-25s  %20llu %8u.%3.3u\n",
                 ct_callp[i].cc_name,
                 ct_callp[i].cc_count,
                 ct_callp[i].cc_avg_wait_whole,
                 ct_callp[i].cc_avg_wait_decimal);

          printf("\n");

          /* Get the pointers to the next system entry */
          p = (char*) ct_sysp;
          p += entry_size;
          ct_sysp = (CT_SYS_STATS*) p;

          p += ct_sysp->cs_length;
          ct_callp = (CT_CALL_STATS*) p;
        }
      }
      else
      {
        /* Second API call failed */
        printf("Error on next request for ctkc stats\n");
        printf("Return Value: %d Return Code: %d Reason Code: %x\n",
               bpxrv, bpxrc, bpxrs);
        buffer_success = -1;
      }
    }
    else
    {
      /* Expecting E2BIG and it was a different error */
      printf("Error on storage stats trying to get required size\n");
```

```
          printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",
                 bpxrv, bpxrc, bpxrs);
          buffer_success = -1;
        }
      }
      else
      {
        /* If rv is 0, most likely there was no data to get */
        if (st_req->sa_len == 0)
        {
          printf("No data\n");
          printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",
                 bpxrv, bpxrc, bpxrs);
          buffer_success = -1;
        }
        else
        {   /* No, there was other problem with getting the size needed */
          printf("Error getting size required\n");
          printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",
                 bpxrv, bpxrc, bpxrs);
          buffer_success = -1;
        }
      }
    }

    if( t == 1000 )
      printf("Number of failed buffer resizes exceeded.\n");

    free(buffp);
    return 0;
  }
```

Statistics Sysplex Owner Operations Information

Purpose

Returns information about the number of calls processed on the local system as a result of a message sent from another system. Vnode operation statistics are returned for each client system that accessed a file system owned on the local server.

Format

```
syscall_parmlist
  opcode              int                 253         STATOP_SVI
  parms[0]            int                 offset to   STAT_API
  parms[1]            int                 Offset of output following STAT_API
  parms[2]            int                 0
  parms[3]            int                 0
  parms[4]            int                 0
  parms[5]            int                 0
  parms[6]            int                 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_supported_ver    int                 version of data returned or reserved
  sa_reserve          int[3]              0
  posix_time_high     unsigned int        high order 32 bits since epoch
  posix_time_low      unsigned int        low order 32 bits since epoch
  posix_useconds      unsigned int        microseconds
  pad1                int

CT_HEADER
  ct_eye              char[4]             "CTHD"
  ct_length           short               Length of the structure
  ct_version          short               Structure version
  number_of_ct_sys    unsigned int        Number of CT_SYS_STATS structures
  number_of_ct_call   unsigned int        Number of CT_CALL_STATS structures

CT_SYS_STATS[number_of_ct_sys]
  cs_eye              char[4]             "CTSY"
  cs_length           short               Length of the structure
  cs_version          short               Structure version
  cs_sysname          char[9]             Name of system.  A value of 0
                                          means there is no information in
                                          this record and any subsequent
                                          record (end of list)
  reserved            char[7]

CT_CALL_STATS[number_of_ct_call]
  cc_eye              char[4]             "CTCL"
  cc_length           short               Length of structure
  cc_version          short               Structure version
  cc_count            unsigned long long  Number of calls of that type
                                          since last statistics reset.
  cc_xcfreq           unsigned long long  Indicates if an XCF request
                                          was required to process the call.
                                          Number of XCF requests that were
                                          required to make callbacks to one
                                          or more clients to process
                                          the requests.
  cc_qwait            unsigned long long  Number of times a request had
                                          to wait in queue before being
                                          dispatched to a processing
                                          task at the owner, valid only
                                          for SVI report
  cc_avg_wait_whole   int                 Average time for system to
                                          process call in milliseconds.
                                          This will be average time for the
                                          owner to process the call for SVI
                                          reports.  This is the part
                                          before the decimal point.
  cc_avg_wait_decimal int                 The part after the decimal
                                          point for avg. waits time.
                                          This is microseconds.
  cc_name             char[25]
  reserved            char[7]

CT_CALL_STATS
  cc_eye              char[4]             "CTCL"
  cc_length           short               Length of structure
  cc_version          short               Structure version
  cc_count            unsigned long long  Number of calls of that type
                                          since last statistics reset.
  cc_xcfreq           unsigned long long  Indicates if an XCF request
                                          was required to process the call.
                                          Number of XCF requests that were
                                          required to make callbacks to one
                                          or more clients to process
                                          the requests.
  cc_qwait            unsigned long long  Number of times a request had
                                          to wait in queue before being
                                          dispatched to a processing
                                          task at the owner, valid only
                                          for SVI report
  cc_avg_wait_whole   int                 Average time for system to
                                          process call in milliseconds.
                                          This will be average time for the
                                          owner to process the call for SVI
                                          reports.  This is the part
                                          before the decimal point.
  cc_avg_wait_decimal int                 The part after the decimal
                                          point for avg. waits time.
                                          This is microseconds.
  cc_name             char[25]
  reserved            char[7]

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

Reason_code
    0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Users of the API supply as input a buffer that contains a `syscall_parmlist` followed by a STAT_API structure, followed by an output buffer.
- Output consists of a CT_HEADER followed by an array of CT_SYS_STATS structures and an array of CT_CALL_STATS structures. The number of elements in each array is returned in `number_of_ct_sys` and `number_of_ct_call` respectively.
- If the output buffer is not large enough to contain all of the output, E2BIG is returned and the required size is placed in `sa_len`. The caller can then try the request again with a larger buffer.
- A CT_SYS_STATS structure is returned only for client systems that sent the local server system messages since the last statistics reset.

Privilege required

None.

Related services

Statistics Server Token Management Information

Statistics Sysplex Client Operations Information

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_STATS 0x40000007
#define STATOP_SVI    254   /* inbound calls from remote clients */
#define E2BIG         145

typedef struct syscall_parmlist_t {
  int opcode;                  /* Operation code to perform */
  int parms[7];                /* Specific to type of operation, */

} syscall_parmlist;

typedef  struct  reset_time {
  unsigned int   posix_time_high;
  unsigned int   posix_time_low;
  unsigned int   posix_usecs;
  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_INIT 0x01
  char    sa_flags;            /* command 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 CT_CALL_STATS_t {
  char  cc_eye[4];
#define CC_EYE "CTCL"
  short cc_length;
  short cc_version;
#define CC_VER_INITIAL 1
  unsigned long long cc_count;
  unsigned long long cc_xcfreq;
  unsigned long long cc_qwait; /* number of waits */
  int   cc_avg_wait_whole;     /* average wait time for */
                               /* calls of this type    */
  int   cc_avg_wait_decimal;
  char  cc_name[25];
  char  reserved[7];
} CT_CALL_STATS;

typedef struct CT_SYS_STATS_t {
  char  cs_eye[4];
#define CS_EYE "CTSY"
  short cs_length;
  short cs_version;
#define CS_VER_INITIAL 1
  char  cs_sysname[9];
  char  reserved[7];
} CT_SYS_STATS;

typedef struct CT_HEADER_t {
  char        ct_eye[4];
#define       CT_EYE "CTHD"
  short       ct_length;
  short       ct_version;
#define       CT_VER_INITIAL 1
  unsigned int number_of_ct_sys;
  unsigned int number_of_ct_call;
} CT_HEADER;

int main(int argc, char** argv)
{
  int  buff_fill_len = 0;
  int  bpxrv, bpxrc, bpxrs;
  char sysname[9];
  int  num_systems;
  int  num_calls;
  int  entry_size;
  int  mypsize;
  int  buflen;
  int  i,j,t;
  int  buffer_success = 0;

  STAT_API          local_req;
  STAT_API*         st_req     = NULL;
  syscall_parmlist* parmp      = NULL;
  CT_HEADER*        ct_p       = NULL;
  CT_SYS_STATS*     ct_sysp    = NULL;
  CT_CALL_STATS*    ct_callp   = NULL;
  char*             p          = NULL;
  char*             buffp      = NULL;

  /* Initialize the local_req to 0s */
  st_req = &local_req;
  memset( st_req, 0x00, sizeof(STAT_API) );

  strcpy( local_req.sa_eye, SA_EYE, sizeof(local_req.sa_eye) );
```

```
        local_req.sa_len = 0;
        local_req.sa_ver = SA_VER_INIT;

        /* Allocate Buffer */
        buffp = (char*) malloc(sizeof(syscall_parmlist) + sizeof(STAT_API));
        if( buffp == NULL )
        {
          printf("Malloc Error\n");
          return 0;
        }
        memset( buffp, 0x00, sizeof(syscall_parmlist) + sizeof(STAT_API));

        /* Set the run parms */
        parmp = (syscall_parmlist*) &buffp[0];
        parmp->opcode   = STATOP_SVI;
        parmp->parms[0] = buff_fill_len = sizeof(syscall_parmlist);
        parmp->parms[1] = buff_fill_len + sizeof(STAT_API);
        parmp->parms[2] = 0;
        parmp->parms[3] = 0;
        parmp->parms[4] = 0;
        parmp->parms[5] = 0;
        parmp->parms[6] = 0;

        st_req = (STAT_API*) &buffp[buff_fill_len];
        memcpy( st_req, &local_req, sizeof(STAT_API) );
        buff_fill_len += sizeof(STAT_API);

        BPX1PCT("ZFS     ",
                ZFSCALL_STATS,              /* Aggregate operation */
                buff_fill_len,              /* Length of Argument */
                (char*) buffp,              /* Pointer to Argument */
                &bpxrv,                     /* Pointer to Return_value */
                &bpxrc,                     /* Pointer to Return_code */
                &bpxrs);                    /* Pointer to Reason_code */

        printf("bpxrv %d\n", bpxrv);

        for(t = 0; t < 1000 && buffer_success == 0; t++)
        {
          if( bpxrv < 0 )
          {
            /* Look for E2BIG to get required file size back in the st_req */
            if( bpxrc == E2BIG )
            {
              buflen  = st_req->sa_len;
              mypsize = sizeof(syscall_parmlist) +
                        sizeof(STAT_API) + buflen;

              free(buffp);

              buffp = (char*) malloc(mypsize);
              if( buffp == NULL )
              {
                printf("Malloc Error\n");
                return 0;
              }
              memset( buffp, 0x00, mypsize );
              printf("Need buffer size of %d, for a total of %d\n",
                     buflen, mypsize);

              /* Set the run parms */
              parmp = (syscall_parmlist*) &buffp[0];
              parmp->opcode   = STATOP_SVI;
              parmp->parms[0] = buff_fill_len = sizeof(syscall_parmlist);
              parmp->parms[1] = buff_fill_len + sizeof(STAT_API);
              parmp->parms[2] = 0;
              parmp->parms[3] = 0;
```

```
      parmp->parms[4] = 0;
      parmp->parms[5] = 0;
      parmp->parms[6] = 0;

      st_req = (STAT_API*) &buffp[buff_fill_len];
      memcpy( st_req->sa_eye, SA_EYE, 4 );
      buff_fill_len += sizeof(STAT_API);
      st_req->sa_ver = SA_VER_INIT;
      st_req->sa_len = buflen;

      BPX1PCT("ZFS     ",
              ZFSCALL_STATS,          /* Aggregate operation */
              mypsize,                /* Length of Argument */
              (char*) buffp,          /* Pointer to Argument */
              &bpxrv,                 /* Pointer to Return_value */
              &bpxrc,                 /* Pointer to Return_code */
              &bpxrs);                /* Pointer to Reason_code */

      if( bpxrv != 0 && bpxrc == E2BIG )
        printf("E2BIG: %d times total\n", t);
      else if( bpxrv == 0 )
      {
        buffer_success = 1;
        ct_p = (CT_HEADER*) &buffp[buff_fill_len];
        buff_fill_len += ct_p->ct_length;
        ct_sysp = (CT_SYS_STATS*) &buffp[buff_fill_len];
        buff_fill_len += ct_sysp->cs_length;
        ct_callp = (CT_CALL_STATS*) &buffp[buff_fill_len];

        /* Make sure there are systems */
        num_systems = ct_p->number_of_ct_sys;
        if( num_systems == 0 )
        {
          printf("Svi stats completed successfully. "
                 "There is no information to display\n");
          free(buffp);
          return 0;
        }
        num_calls  = ct_p->number_of_ct_call;
        entry_size = ct_sysp->cs_length +
          (ct_callp->cc_length * num_calls);

        for (j = 0; j < num_systems; j++)
        {
          printf("SV");
          printf("%30cSVI Calls from System %s\n", ' ',
                 ct_sysp->cs_sysname);
          printf("  ");
          printf("%30c------------------------------\n", ' ');
          printf("SVI Call                 "
                 "Count                "
                 "Qwait            "
                 "XCF Req.         "
                 "Avg. Time\n");
          printf("--------------------     "
                 "--------------------  "
                 "----------------  ----------------  "
                 "----------\n");

          for (i = 0; i < num_calls-1; i++)
          {
            printf("%-25s%20llu  %16llu  %16llu%8u.%3.3u\n",
                   ct_callp[i].cc_name,
                   ct_callp[i].cc_count,
                   ct_callp[i].cc_qwait,
                   ct_callp[i].cc_xcfreq,
                   ct_callp[i].cc_avg_wait_whole,
                   ct_callp[i].cc_avg_wait_decimal);
          }

          /* Put out the Totals entry */
          printf("--------------------     "
                 "--------------------  "
                 "----------------  ----------------  "
                 "----------\n");
          printf("%-25s%20llu  %16llu  %16llu%8u.%3.3u\n",
                 ct_callp[i].cc_name,
                 ct_callp[i].cc_count,
                 ct_callp[i].cc_qwait,
                 ct_callp[i].cc_xcfreq,
                 ct_callp[i].cc_avg_wait_whole,
                 ct_callp[i].cc_avg_wait_decimal);

          printf("\n");

          /* Get the pointers to the next system entry */
          p = (char*) ct_sysp;
          p += entry_size;
          ct_sysp = (CT_SYS_STATS*) p;

          p += ct_sysp->cs_length;
          ct_callp = (CT_CALL_STATS*) p;
        }
      }
      else
      {
        /* Second API call failed */
        printf("Error on next request for svi stats\n");
        printf("Return Value: %d "
               "Return Code: %d "
               "Reason Code: %x\n",
               bpxrv, bpxrc, bpxrs);
        buffer_success = -1;
      }
    }
    else
    {
      /* Expecting E2BIG and it was a different error */
      printf("Error on storage stats trying to get required size\n");
      printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",
             bpxrv, bpxrc, bpxrs);
      buffer_success = -1;
    }
  }
  else
  {
    /* If rv is 0, most likely there was no data to get */
    if (st_req->sa_len == 0)
    {
      printf("No data\n");
      printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",
             bpxrv, bpxrc, bpxrs);
    }
    else
    {   /* There was some other problem with getting required size */
      printf("Error getting size required\n");
      printf("BPXRV = %d BPXRC = %d BPXRS = %x\n",
             bpxrv, bpxrc, bpxrs);
    }
    buffer_success = -1;
  }
}

if( t == 1000 )
```

```
      printf("Number of failed buffer resizes exceeded.\n");

    free(buffp);
    return 0;
  }
```

Statistics Transaction Cache Information

Purpose

A performance statistics operation that returns transaction cache counters. It determines the number of transactions in the transaction cache.

| As of z/OS V2R2, this subcommand is no longer used. All output fields from a call
| to statistics transaction cache information will be filled in with zeros.

Format

```
syscall_parmlist
   opcode                int              250   STATOP_TRAN_CACHE
   parms[0]              int              Offset to STAT_API
   parms[1]              int              Offset of output following STAT_API
   parms[2]              int              Offset to system name (optional)
   parms[3]              int              0
   parms[4]              int              0
   parms[5]              int              0
   parms[6]              int              0
STAT_API
   sa_eye                char[4]          "STAP"
   sa_len                int              Length of buffer following STAT_API
   sa_ver                int              1
   sa_flags              char[1]          0x80 - Reset statistics
   sa_fill               char[3]          Reserved
   sa_reserve            int[4]           Reserved
   posix_time_high       unsigned int     High order 32 bits since epoch
   posix_time_low        unsigned int     Low order 32 bits since epoch
   posix_useconds        unsigned int     Microseconds
   pad1                  int              Reserved
STAT_TRAN_CACHE
   sttr_started_high     unsigned int     Transactions started high 32 bits
   sttr_started          unsigned int     Transactions started
   sttr_lookups_high     unsigned int     Lookups on transaction high 32
                                          bits
   sttr_lookups          unsigned int     Lookups on transaction
   sttr_ec_merges_high   unsigned int     Equivalence class merges high 32
                                          bits
   sttr_ec_merges        unsigned int     Equivalence class merges
   sttr_alloc_trans_high unsigned int     Allocated transactions high 32
                                          bits
   sttr_alloc_trans      unsigned int     Allocated transactions
   sttr_trans_act_high   unsigned int     Transactions active high 32 bits
   sttr_trans_act        unsigned int     Transactions active
   sttr_trans_pend_high  unsigned int     Transactions pending high 32 bits
   sttr_trans_pend       unsigned int     Transactions pending
   sttr_trans_comp_high  unsigned int     Transactions completed high 32
                                          bits
   sttr_trans_comp       unsigned int     Transactions completed
   sttr_trans_free_high  unsigned int     Free transactions high 32 bits
   sttr_trans_free       unsigned int     Free transactions
   reserved              char[60]         Reserved
systemname               char[9]          System name to get stas from
```

```
Return_value     0 if request is successful, -1 if it is not successful

Return_code
  EINTR          zFS is shutting down
  EINVAL         Invalid parameter list
  EMVSERR        Internal error occurred
  E2BIG          Information too big for buffer supplied

Reason_code
  0xEFnnxxxx     See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

None.

Related services

Statistics 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_TRAN_CACHE 250 /* 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 stat_tran_cache_t
{
    unsigned int      sttr_started_high;
    unsigned int      sttr_started;
    unsigned int      sttr_lookups_high;
    unsigned int      sttr_lookups;
    unsigned int      sttr_ec_merges_high;
    unsigned int      sttr_ec_merges;
    unsigned int      sttr_alloc_trans_high;
    unsigned int      sttr_alloc_trans;
    unsigned int      sttr_trans_act_high;
    unsigned int      sttr_trans_act;
    unsigned int      sttr_trans_pend_high;
    unsigned int      sttr_trans_pend;
    unsigned int      sttr_trans_comp_high;
    unsigned int      sttr_trans_comp;
    unsigned int      sttr_trans_free_high;
    unsigned int      sttr_trans_free;
    char              reserved[60];
} STAT_TRAN_CACHE;

/* reset timestamp */
typedef struct reset_time {
    unsigned int      posix_time_high; /* high order 32 bits since epoc */
    unsigned int      posix_time_low;  /* low order 32 bits since epoch */
    unsigned int      posix_usecs;     /* microseconds */
    int               pad1;
} RESET_TIME;

/*********************************************************************/
/* The following structure is the api query control block.          */
/* It is used for all api query commands.                            */
/*********************************************************************/
typedef struct stat_api_t
{
#define SA_EYE "STAP"
    char              sa_eye[4];     /* 4 byte identifier must be */
    int               sa_len;        /* length of the buffer to put data into*/
                                     /* this buffer area follows this struct*/
    int               sa_ver;        /* the version number currently always 1*/
#define               SA_VER_INITIAL 0x01
    char              sa_flags;      /* flags field must be x00 or x80,
                                        x80 means reset statistics*/
#define               SA_RESET 0x80
    char              sa_fill[3];    /* spare bytes */
    int               sa_reserve[4]; /* Reserved */
    struct reset_time reset_time_info;
} STAT_API;

struct parmstruct {
    syscall_parmlist myparms;
    STAT_API         myapi;
    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;

    /* Only specify a non-zero offset for the next field (parms[2]) if */
    /* you want to query the tran cache statistics of another system.  */
    /* myparmstruct.myparms.parms[2] = sizeof(syscall_parmlist) +      */
    /*                                 sizeof(STAT_API) +              */
    /*                                 sizeof(STAT_TRAN_CACHE);        */

    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    memset(stapptr, 0, sizeof(STAT_API));
    memcpy(stapptr->sa_eye, SA_EYE, 4);
    stapptr->sa_ver = SA_VER_INITIAL;
    stapptr->sa_len = (int) sizeof(STAT_TRAN_CACHE);

    /* This next field should only be set if parms[2] is non-zero */
    /* strcpy(myparmstruct.systemname,"DCEIMGVQ"); */

    BPX1PCT("ZFS     ",
            ZFSCALL_STATS,              /* Perf statistics operation */
            sizeof(myparmstruct),       /* Length of Argument */
            (char *)&myparmstruct,      /* Pointer to Argument */
            &bpxrv,                     /* Pointer to Return_value */
            &bpxrc,                     /* Pointer to Return_code */
            &bpxrs);                    /* Pointer to Reason_code */

    if (bpxrv < 0)
    {
        printf("Error querying tran cache, BPXRV = %d BPXRC = %d BPXRS = %x\n",
               bpxrv, bpxrc, bpxrs);
        return bpxrc;
    }
    else
    {
        printf("\n%52s\n", "Transaction Cache Statistics");
        printf("%52s\n", "----------------------------");
        printf("Trans started: %8u Lookups on Tran: %8u EC Merges: %8u\n",
               myparmstruct.mystats.sttr_started,
               myparmstruct.mystats.sttr_lookups,
               myparmstruct.mystats.sttr_ec_merges);

        printf("Allocated Trans: %8u \n(Act= %7u, Pend= %7u, ",
               myparmstruct.mystats.sttr_alloc_trans,
               myparmstruct.mystats.sttr_trans_act,
               myparmstruct.mystats.sttr_trans_pend);

        printf("Comp=%7u, Free= %7u)\n",
               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] = ' ';
            buf[19] = '.';
            printf("Last Reset Time: %s\n", buf);
        }
    }
    return 0;
}
```

Statistics User Cache Information

Purpose

A performance statistics operation that returns user cache information.

Format

```
syscall_parmlist
   opcode              int             242    STATOP_USER_CACHE
   parm[0]             int             Offset to STAT_API
   parm[1]             int             Offset of output following STAT_API
   parm[2]             int             Offset to system name (optional)
   parm[3]             int             0
   parm[4]             int             0
   parm[5]             int             0
   parm[6]             int             0
STAT_API
   sa_eye              char[4]         "STAP"
   sa_len              int             Length of buffer that follows STAT_API
   sa_ver              int             1 or 2
   sa_flags            char[1]         0x80 for reset; 0x00 otherwise
   sa_fill             char[3]         Reserved
   sa_supported_ver    int             Version of data returned when sa_ver
                                       is 2
   sa_reserve          int[3]          Reserved
   posix_time_high     unsigned int    High order 32 bits since epoch
   posix_time_low      unsigned int    Low order 32 bits since epoch
   posix_useconds      unsigned int    Microseconds
   pad1                int             Reserved

STAT_USER_CACHE[2]
   VM_STATS[2]
       vm_schedules          unsigned int      Number of I/O requests
       vm_setattrs           unsigned int      Number of setattr requests
       vm_fsyncs             unsigned int      Number of fsync operations
       vm_unmaps             unsigned int      Number of file deletions
       vm_reads              unsigned int      Number of read operations
       vm_readasyncs         unsigned int      Number of readaheads
       vm_writes             unsigned int      Number of write operations
       vm_getattrs           unsigned int      Number of getattr requests
       vm_flushes            unsigned int      Number of cache flushes
       vm_scheduled_deletes  unsigned int      Number of times an I/O is
                                               canceled because the file was
                                               deleted
       vm_reads_faulted      unsigned int      Number of times I/O needed to
                                               satisfy read operation
                                               (data was not in cache)
       vm_writes_faulted     unsigned int      Number of times I/O needed to
                                               read data before data can be
                                               written to cache
       vm_read_ios           unsigned int      Total number of file system
                                               reads for any reason
       vm_scheduled_writes   unsigned int      Number of data write I/Os issued
       vm_error_writes       unsigned int      Number of data writes done when
                                               flushing a file from the cache
                                               after an I/O error or canceled
                                               user
```

```
      vm_reclaim_writes     unsigned int      Number of data writes during
                                              space reclaim
      vm_read_waits         unsigned int      Number of times a read had to
                                              wait for pending I/O
      vm_write_waits        unsigned int      Number of waits for pending I/O
                                              so that new data could be
                                              written to the file
      vm_fsync_waits        unsigned int      Number of waits for pending I/O
                                              fsync operations did
      vm_error_waits        unsigned int      Number of waits when flushing a
                                              file from the cache after an I/O
                                              error or canceled user
      vm_reclaim_waits      unsigned int      Number of waits done during
                                              reclaim processing for I/O
      vm_reclaim_steal      unsigned int      Number of pages stolen during
                                              space reclaim processing
      vm_waits_for_reclaim  unsigned int      Number of waits for reclaim
                                              processing to complete
      vm_reserved           int[10]           Reserved
   suc dataspaces           int               Number of dataspaces in user
                                              data cache
   suc pages_per_dataspace  int               Number of pages per dataspace
   suc seg_size_local       int               Local segment size (in K)
   suc seg_size_remote      int               Remote segment size (in K)
   suc page_size            int               Page size (in K)
   suc cache_pages          int               Number of pages in user cache
   suc total_free           int               Number of free pages
   suc segment_cachesize    int               Number of segments
   stuc_reserved            int[5]            Reserved
   DS_ENTRY[32]
      ds_name               char[9]           Dataspace name
      pad1                  char[3]           Reserved
      ds_alloc_segs         int               Number of used (allocated)
                                              segments in the dataspace
      ds_free_pages         int               Number of free dataspace pages
      ds_reserved           int[5]            Reserved
STAT_USER_CACHE2
   VM_STATS2
     vm_schedules           unsigned long long int  Number of I/O requests
     vm_setattrs            unsigned long long int  Number of setattrs
     vm_fsyncs              unsigned long long int  Number of fysnc operations
     vm_unmaps              unsigned long long int  Number of file deletions
     vm_reads               unsigned long long int  Number of read operations
     vm_readasyncs          unsigned long long int  Number of readaheads
     vm_writes              unsigned long long int  Number of write operations
     vm_getattrs            unsigned long long int  Number of getattrs
     vm_flushes             unsigned long long int  Number of times the
                                                    user cache was flushed

     vm_scheduled_deletes   unsigned long long int  Number of times an I/O
                                                    is canceled because the
                                                    file was deleted
      vm_reads_faulted       unsigned long long int  Number of times I/O needed
                                                     to satisfy read operation
                                                     (data was not in cache)
      vm_writes_faulted      unsigned long long int  Number of times I/O needed
                                                     to read data before data
                                                     can be written to cache
      vm_read_ios            unsigned long long int  Total number of file
                                                     system reads for any
                                                     reason
      vm_scheduled_writes    unsigned long long int  Number of data write
                                                     I/Os issued
      vm_error_writes        unsigned long long int  Number of data writes
                                                     when flushing a file
                                                     from the cache after
                                                     an I/O error or a
                                                     canceled user
      vm_reclaim_writes      unsigned long long int  Number of data writes
                                                     during space reclaim
      vm_read_waits          unsigned long long int  Number of times a read had
                                                     to wait for pending I/O
      vm_write_waits         unsigned long long int  Number of waits for a
                                                     pending I/O so that new
                                                     data could be written to
                                                     the file
      vm_fsync_waits         unsigned long long int  Number of waits for
                                                     pending I/O fsync
                                                     operations did
      vm_error_waits         unsigned long long int  Number of waits in
                                                     user cache error
                                                     processing
      vm_reclaim_waits       unsigned long long int  Number of waits done
                                                     during the reclaim
                                                     processing for I/O
      vm_reclaim_steal       unsigned long long int  Number of user cache
                                                     pages stolen during
                                                     reclaim processing
      vm_waits_for_reclaim   unsigned long long int  Number of waits for
                                                     space reclaim process
                                                     to complete
      vm_reserved            unsigned long long int[10]  Reserved
    suc dataspaces           int                     Number of dataspaces in
                                                     user data cache
    suc pages_per_dataspace  int                     Number of pages per
                                                     dataspace
    suc seg_size_local       int                     Local segment size (in K)
    suc seg_size_remote      int                     Remote segment size (in K)
    suc page_size            int                     Page size (in K)
    suc cache_pages          int                     Number of pages in cache
    suc total_free           int                     Number of free pages
    suc segment_cachesize    int                     Number of segments
    stuc_reserved            int[5]                  Reserved
    DS_ENTRY[32]
      ds_name                char[9]                 Dataspace name
      pad1                   char[3]                 Reserved
      ds_alloc_segs          int                     Number of used segments in
                                                     dataspace
      ds_free_pages          int                     Number of free pages in
                                                     dataspace
      ds_reserved             int[5]                 Reserved
   reserved                  char[4]                 Reserved
systemname                   char[9]                 Name of system to get
                                                     statistics from
```

```
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
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.
- When `sa_supported_ver` is 0 or 1, the output consists of STAT_USER_CACHE[2] and DS_ENTRY.
- When `sa_supported_ver` is 2 the output consists of STAT_USER_CACHE2 and DS_ENTRY.

Privilege required

None.

Related services

Statistics Locking Information
Statistics Storage Information

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>

#define ZFSCALL_STATS     0x40000007
#define STATOP_USER_CACHE 242 /* Performance API queries */
#define NUM_DATASPACES    32
#define REMOTE            1
#define LOCAL             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 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 {
  unsigned int      posix_time_high; /* high order 32 bits since epoc */
  unsigned int      posix_time_low;  /* low order 32 bits since epoch */
  unsigned int      posix_usecs;     /* microseconds */
  int               pad1;
} RESET_TIME;

/*********************************************************************/
/* The following structure is the user data cache statistics */
/*********************************************************************/
typedef struct vm_stats_2_t
{
  /**********************************************************************/
  /* First set of counters are for external requests to the VM system.  */
  /**********************************************************************/
  unsigned long long int vm_schedules;
  unsigned long long int vm_setattrs;
  unsigned long long int vm_fsyncs;
  unsigned long long int vm_unmaps;
  unsigned long long int vm_reads;
  unsigned long long int vm_readasyncs;
  unsigned long long int vm_writes;
  unsigned long long int vm_getattrs;
  unsigned long long int vm_flushes;
  unsigned long long int 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).                          */
  /**********************************************************************/
  unsigned long long int vm_reads_faulted;
  unsigned long long int vm_writes_faulted;
  unsigned long long int vm_read_ios;

  /**********************************************************************/
  /* Next counters are write counters.  They measure number of times    */
  /* we scheduled and waited for write I/Os.                            */
  /**********************************************************************/
  unsigned long long int vm_scheduled_writes;
  unsigned long long int vm_error_writes;
  unsigned long long int 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.    */
  /**********************************************************************/
  unsigned long long int vm_read_waits;
  unsigned long long int vm_write_waits;
  unsigned long long int vm_fsync_waits;
  unsigned long long int vm_error_waits;
  unsigned long long int vm_reclaim_waits; /* Waited for pending
                                              I/O for reclaim */

  /**********************************************************************/
  /* Final set are memory management counters.                          */
  /**********************************************************************/
  unsigned long long int vm_reclaim_steal;       /* Number of times steal from
                                                    others function invoked */
  unsigned long long int vm_waits_for_reclaim;   /* Waits for reclaim thread */
  unsigned long long int vm_reserved[10];        /*reserved for future use*/
} VM_STATS_2;

typedef struct stat_user_cache_2_t
{
   /*Various statistics for both LOCAL and REMOTE systems */
  VM_STATS_2 stuc;

  int       stuc_dataspaces;     /* Number of dataspaces in user data cache */
  int       stuc_pages_per_ds;   /* Pages per dataspace */
  int       stuc_seg_size_loc;   /* Local Segment Size (in K) */
  int       stuc_seg_size_rmt;   /* Remote Segment Size (in K) */
  int       stuc_page_size;      /* Page Size (in K) */
  int       stuc_cache_pages;    /* Total number of pages */
  int       stuc_total_free;     /* Total number of free pages */
  int       stuc_vmSegTable_cachesize; /* Number of segments */
  int       stuc_reserved[5];          /*reserved for future use*/
  DS_ENTRY  stuc_ds_entry[NUM_DATASPACES];  /* Array of dataspace entries */
  char      reserved[4];
} STAT_USER_CACHE_2;

/* Version 1 Output Structures */

/*********************************************************************/
/* 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. */
  /**********************************************************************/
  unsigned int           vm_schedules;
  unsigned int           vm_setattrs;
  unsigned int           vm_fsyncs;
  unsigned int           vm_unmaps;
  unsigned int           vm_reads;
  unsigned int           vm_readasyncs;
  unsigned int           vm_writes;
  unsigned int           vm_getattrs;
  unsigned int           vm_flushes;
  unsigned int           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).                           */
  /**********************************************************************/
  unsigned int           vm_reads_faulted;
  unsigned int           vm_writes_faulted;
  unsigned int           vm_read_ios;
  /**********************************************************************/
  /* Next counters are write counters. They measure number of times     */
  /* we scheduled and waited for write I/Os.                            */
  /**********************************************************************/
  unsigned int           vm_scheduled_writes;
  unsigned int           vm_error_writes;
  unsigned int           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.    */
  /**********************************************************************/
  unsigned int           vm_read_waits;
  unsigned int           vm_write_waits;
  unsigned int           vm_fsync_waits;
  unsigned int           vm_error_waits;
  unsigned int           vm_reclaim_waits; /* Waited for pending
                                              I/O for reclaim */

  /**********************************************************************/
  /* Final set are memory management counters.                          */
  /**********************************************************************/
  unsigned int           vm_reclaim_steal;     /* Number of times steal from
                                                  others function invoked */
  unsigned int           vm_waits_for_reclaim; /* Waits for reclaim thread */
  unsigned int           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 dataspace */
  int               stuc_seg_size_loc;         /* Local Segment Size (in K) */
  int               stuc_seg_size_rmt;         /* Remote Segment Size (in K) */
  int               stuc_page_size;            /* Page Size (in K) */
  int               stuc_cache_pages;          /* Total number of pages */
  int               stuc_total_free;           /* Total number of free pages */
  int               stuc_vmSegTable_cachesize; /* Number of segments */
  int               stuc_reserved[5];          /* reserved */
  DS_ENTRY          stuc_ds_entry[32];         /* Array of dataspace entries */
} STAT_USER_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*/
                                   /* this buffer area follows this struct */
  int               sa_ver;        /* the version number currently always 1*/
#define             SA_VER_2 0x02
#define             SA_VER_INIT 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_supported_ver; /* version of data returned */
  int               sa_reserve[3];    /* Reserved */
  struct reset_time reset_time_info;
} STAT_API;

struct parmstruct {
  syscall_parmlist  myparms;
  STAT_API          myapi;
  STAT_USER_CACHE_2 mystats;
  char              systemname[9];
} myparmstruct;

int print_user_cache_version1(STAT_USER_CACHE   *stcacheptr);
int print_user_cache_version2(STAT_USER_CACHE_2 *stcacheptr);

int main(int argc, char **argv)
{
  int      bpxrv;
  int      bpxrc;
  int      bpxrs;
  int      i,j;
  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_2);  */

  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_2;
  stapptr->sa_len = (int) sizeof(STAT_USER_CACHE_2);

  /* 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
  {
    if( stapptr->sa_supported_ver == SA_VER_INIT )
    {
      STAT_USER_CACHE *stcacheptr_v1;
      stcacheptr_v1 = (STAT_USER_CACHE*) &(myparmstruct.mystats);
      print_user_cache_version1(stcacheptr_v1);
    }
    else
    {
      STAT_USER_CACHE_2 *stcacheptr = &(myparmstruct.mystats);
      print_user_cache_version2(stcacheptr);
    }

    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;
}

int print_user_cache_version2(STAT_USER_CACHE_2* stcacheptr)
{
  int i;
  double   ratio1, ratio2, ratio3, ratio4;
  printf("                   User File (VM) Caching System Statistics\n");
  printf("                   ----------------------------------------\n");
  printf("\n");

  printf("                   Direct Statistics\n");
  printf("                   -----------------\n\n");

  printf("External Requests:\n");
  printf("------------------\n");
  printf("%-9s %20llu     %-9s %20llu     %-9s %20llu\n",
         "Reads"    , stcacheptr->stuc.vm_reads,
         "Fsyncs"   , stcacheptr->stuc.vm_fsyncs,
         "Schedules", stcacheptr->stuc.vm_schedules);
  printf("%-9s %20llu     %-9s %20llu     %-9s %20llu\n",
         "Writes"   , stcacheptr->stuc.vm_writes,
         "Setattrs" , stcacheptr->stuc.vm_setattrs,
         "Unmaps"   , stcacheptr->stuc.vm_unmaps);
  printf("%-9s %20llu     %-9s %20llu     %-9s %20llu\n",
         "Asy Reads", stcacheptr->stuc.vm_readasyncs,
         "Getattrs" , stcacheptr->stuc.vm_getattrs,
         "Flushes"  , stcacheptr->stuc.vm_flushes);
  printf("\n");

  printf("File System Reads:\n");
  printf("------------------\n");

  ratio1 = ratio2 = ratio3 = ratio4 = 0.0;

  if (stcacheptr->stuc.vm_reads > 0)
  {
    ratio1 = 100 * (((double)stcacheptr->stuc.vm_reads_faulted)
                    / ((double)stcacheptr->stuc.vm_reads));
  }
  if (stcacheptr->stuc.vm_writes > 0)
  {
    ratio2 = 100 * (((double)stcacheptr->stuc.vm_writes_faulted)
                    / ((double)stcacheptr->stuc.vm_writes));
  }
  if (stcacheptr->stuc.vm_reads > 0)
  {
    ratio3 = 100 * (((double)stcacheptr->stuc.vm_read_waits)
                    / ((double)stcacheptr->stuc.vm_reads));
  }

  printf("%-14s %20llu (%s Ratio %.2f%%)\n",
         "Reads Faulted", stcacheptr->stuc.vm_reads_faulted,
         "Fault", ratio1);

  printf("%-14s %20llu (%s Ratio %.2f%%)\n",
         "Writes Faulted", stcacheptr->stuc.vm_writes_faulted,
         "Fault", ratio2);

  printf("%-14s %20llu (%s Ratio %.2f%%)\n",
         "Read Waits", stcacheptr->stuc.vm_read_ios,
         "Wait", ratio3);

  printf("\n");
  printf("File System Writes:\n");
  printf("-------------------\n");
  printf("%-19s %20llu %-13s %20llu\n",
         "Scheduled Writes"   ,stcacheptr->stuc.vm_scheduled_writes,
         "Sync Waits"         ,stcacheptr->stuc.vm_fsync_waits);

  printf("%-19s %20llu %-13s %20llu\n",
         "Error Writes"       ,stcacheptr->stuc.vm_error_writes,
         "Error Waits"        ,stcacheptr->stuc.vm_error_waits);

  printf("%-19s %20llu %-13s %20llu\n",
        "Page Reclaim Writes", stcacheptr->stuc.vm_reclaim_writes,
        "Reclaim Waits"      , stcacheptr->stuc.vm_reclaim_waits);

  if (stcacheptr->stuc.vm_writes > 0)
  {
    ratio4 = 100 * (((double)stcacheptr->stuc.vm_write_waits)
                   / ((double)stcacheptr->stuc.vm_writes));
  }
  printf("%-19s %20llu (Wait Ratio %.2f%%)\n",
         "Write Waits", stcacheptr->stuc.vm_write_waits,
         ratio4);


  printf("\n");
  printf("Page Management (Segment Size = (%dK Local %dK Remote) ) "
         "(Page Size = %dK)\n",
         stcacheptr->stuc_seg_size_loc,
         stcacheptr->stuc_seg_size_rmt,
         stcacheptr->stuc_page_size);
  printf("----------------------------------------"
         "---------------------------------\n");

  printf("Total Pages       %10u     Free               %10u\n",
         stcacheptr->stuc_cache_pages,
         stcacheptr->stuc_total_free);
  printf("Segments          %10u\n",
         stcacheptr->stuc_vmSegTable_cachesize);
  printf("Steal Invocations %20llu     Waits for Reclaim %21llu\n\n",
         stcacheptr->stuc.vm_reclaim_steal,
         stcacheptr->stuc.vm_waits_for_reclaim);

  printf("Number of dataspaces used: %5d ",
         stcacheptr->stuc_dataspaces);
  printf("Pages per dataspace: %11d\n",
         stcacheptr->stuc_pages_per_ds);
  printf("\n");
  printf("Dataspace   Allocated     Free\n");
  printf("Name        Segments      Pages\n");
  printf("--------    ----------    ----------\n");

  for (i = 0; i < stcacheptr->stuc_dataspaces; i++)
  {
    printf("%8s    %10u    %10u\n\n",
           stcacheptr->stuc_ds_entry[i].ds_name,
           stcacheptr->stuc_ds_entry[i].ds_alloc_segs,
           stcacheptr->stuc_ds_entry[i].ds_free_pages);
  }
  return 1;
}

int print_user_cache_version1(STAT_USER_CACHE *stcacheptr)
{
  int i;
  double ratio1, ratio2, ratio3, ratio4;
  printf("Version 1 Output is being displayed\n\n");

  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\n");
    }
```

```
else
{
  printf("\n                  Client Statistics\n");
  printf("                  -----------------\n\n");
}

printf("External Requests:\n");
printf("------------------\n");
printf("%-9s %10u     %-9s %10u     %-9s %10u\n",
       "Reads"    , stcacheptr->stuc[i].vm_reads,
       "Fsyncs"   , stcacheptr->stuc[i].vm_fsyncs,
       "Schedules", stcacheptr->stuc[i].vm_schedules);
printf("%-9s %10u     %-9s %10u     %-9s %10u\n",
       "Writes"   , stcacheptr->stuc[i].vm_writes,
       "Setattrs" , stcacheptr->stuc[i].vm_setattrs,
       "Unmaps"   , stcacheptr->stuc[i].vm_unmaps);
printf("%-9s %10u     %-9s %10u     %-9s %10u\n",
       "Asy Reads", stcacheptr->stuc[i].vm_readasyncs,
       "Getattrs" , stcacheptr->stuc[i].vm_getattrs,
       "Flushes"  , stcacheptr->stuc[i].vm_flushes);
printf("\n");

printf("File System Reads:\n");
printf("------------------\n");

ratio1 = ratio2 = ratio3 = ratio4 = 0.0;

if (stcacheptr->stuc[i].vm_reads > 0)
{
  ratio1 = 100 * (((double)stcacheptr->stuc[i].vm_reads_faulted)
                 / ((double)stcacheptr->stuc[i].vm_reads));
}
if (stcacheptr->stuc[i].vm_writes > 0)
{
  ratio2 = 100 * (((double)stcacheptr->stuc[i].vm_writes_faulted)
                 / ((double)stcacheptr->stuc[i].vm_writes));
}
if (stcacheptr->stuc[i].vm_reads > 0)
{
  ratio3 = 100 * (((double)stcacheptr->stuc[i].vm_read_waits)
                 / ((double)stcacheptr->stuc[i].vm_reads));
}

printf("%-14s %10u (%s Ratio %.2f%%)\n",
       "Reads Faulted", stcacheptr->stuc[i].vm_reads_faulted,
       "Fault", ratio1);

printf("%-14s %10u (%s Ratio %.2f%%)\n",
       "Writes Faulted", stcacheptr->stuc[i].vm_writes_faulted,
       "Fault", ratio2);

printf("%-14s %10u (%s Ratio %.2f%%)\n",
       "Read Waits", stcacheptr->stuc[i].vm_read_ios,
       "Wait", ratio3);

printf("\n");
printf("File System Writes:\n");
printf("-------------------\n");
printf("%-19s %10u %-13s %10u\n",
       "Scheduled Writes"   , stcacheptr->stuc[i].vm_scheduled_writes,
       "Sync Waits"         , stcacheptr->stuc[i].vm_fsync_waits);

printf("%-19s %10u %-13s %10u\n",
       "Error Writes"       , stcacheptr->stuc[i].vm_error_writes,
       "Error Waits"        , stcacheptr->stuc[i].vm_error_waits);

printf("%-19s %10u %-13s %10u\n",
          "Page Reclaim Writes", stcacheptr->stuc[i].vm_reclaim_writes,
          "Reclaim Waits"      , stcacheptr->stuc[i].vm_reclaim_waits);

    if (stcacheptr->stuc[i].vm_writes > 0)
    {
      ratio4 = 100 * (((double)stcacheptr->stuc[i].vm_write_waits)
                      / ((double)stcacheptr->stuc[i].vm_writes));
    }
    printf("%-19s %10u (Wait Ratio %.2f%%)\n",
          "Write Waits", stcacheptr->stuc[i].vm_write_waits,
          ratio4);
  }

  printf("\n");
  printf("Page Management (Segment Size = (%dK Local %dK Remote) ) "
         "(Page Size = %dK)\n",
         stcacheptr->stuc_seg_size_loc,
         stcacheptr->stuc_seg_size_rmt,
         stcacheptr->stuc_page_size);
  printf("--------------------------------------"
         "-----------------------------------\n");

  printf("Total Pages       %10u     Free               %10u\n",
         stcacheptr->stuc_cache_pages,stcacheptr->stuc_total_free);
  printf("Segments          %10u\n",
         stcacheptr->stuc_vmSegTable_cachesize);
  printf("Steal Invocations %10u     Waits for Reclaim %11u\n\n",
         stcacheptr->stuc[0].vm_reclaim_steal,
         stcacheptr->stuc[0].vm_waits_for_reclaim);

  printf("Number of dataspaces used: %5d ", stcacheptr->stuc_dataspaces);
  printf("Pages per dataspace: %11d\n", stcacheptr->stuc_pages_per_ds);
  printf("\n");
  printf("Dataspace   Allocated     Free\n");
  printf("Name        Segments      Pages\n");
  printf("--------    ----------    ----------\n");

  for (i = 0; i < stcacheptr->stuc_dataspaces; i++)
  {
    printf("%8s    %10u    %10u\n\n",
          stcacheptr->stuc_ds_entry[i].ds_name,
          stcacheptr->stuc_ds_entry[i].ds_alloc_segs,
          stcacheptr->stuc_ds_entry[i].ds_free_pages);
  }

  return 1;
}
```

Statistics Vnode Cache Information

Purpose

A performance statistics operation that returns vnode cache counters. It determines the number of requests, hits, and discards from the vnode cache.

Format

```
syscall_parmlist
   opcode                int              251    STATOP_VNODE_CACHE
   parms[0]              int              Offset to STAT_API
   parms[1]              int              Offset of output following STAT_API
   parms[2]              int              Offset to system name (optional)
   parms[3]              int              0
   parms[4]              int              0
   parms[5]              int              0
   parms[6]              int              0
STAT_API
   sa_eye                char[4]          "STAP"
   sa_len                int              Length of buffer that follows
                                          the STAT_API
   sa_ver                int              1 or 2
   sa_flags              char[1]          0x80 - Reset statistics
   sa_fill               char[3]          Reserved
   sa_supported_ver      int              Version of data returned
   sa_reserve            int[3]           Reserved
   posix_time_high       unsigned int     High order 32 bits since epoch
   posix_time_low        unsigned int     Low order 32 bits since epoch
   posix_useconds        unsigned int     Microseconds
   pad1                  int              Reserved
STAT_VNODE_CACHE
  VNM_STATS_API_STRUCT
     reserved            unsigned int     Reserved
     Vnodes              unsigned int     Number of vnodes
     Requests            unsigned int     Number of requests
     Hits                unsigned int     Number of hits
     RatioWhole          hyper            Ratio of hits to requests
                                          (whole number part)
     RatioDecimal        hyper            Ratio of hits to requests
                                          (decimal part).  Decimal part is
                                          in thousanths (3 means .003 and
                                          300 means .3)
     Allocates           hyper            Allocates
     Deletes             hyper            Deletes
     VnodeStructSize     hyper            Base vnode structure size
     ExtendedVnodes      hyper            Number of extended vnodes
     extensionSize       hyper            Size of vnode extension
     USSHeldVnodes       hyper            Number of held vnodes
     USSHeldVnodesHi     hyper            Held vnodes high water mark
     OpenVnodes          hyper            Number of open vnodes
     OpenVnodesHi        hyper            Open vnodes high water mark
     OpenVnodesReuse     hyper            Number vnodes that can be reused
     reserved2           hyper[12]        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
     convert_namecount    unsigned int     Count of names processed during
                                           conversion
     reserved             int              Reserved
     reserved1            hyper[11]        Reserved
  ZFSVNODEOPCOUNTS[50]
     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

-- or --
STAT_VNODE_CACHE2
  VNM_STATS_API_STRUCT2
     reserved            unsigned long long int    Reserved
     Vnodes              unsigned long long int    Number of vnodes
     Requests            unsigned long long int    Number of requests
     Hits                unsigned long long int    Number of hits
     RatioWhole          hyper                     Ratio of hits to requests
                                                   (whole number part)
     RatioDecimal        hyper                     Ratio of hits to requests
                                                   (decimal part).  Decimal
                                                   part is in thousandths
                                                   (3 means .003, 300 is .3)
     Allocates           unsigned long long int    Allocates
     Deletes             unsigned long long int    Deletes
     VnodeStructSize     unsigned long long int    Base vnode structure size
     ExtendedVnodes      unsigned long long int    Number of extended vnodes
     extensionSize       unsigned long long int    Size of vnode extension
     USSHeldVnodes       unsigned long long int    Number of held vnodes
     USSHeldVnodesHi     unsigned long long int    Held vnode high water mark
     OpenVnodes          unsigned long long int    Number of open vnodes
     OpenVnodesHi        unsigned long long int    Open vnode high water mark
     OpenVnodesReuse     unsigned long long int    Number of vnodes that can
                                                   be reused
     extCleans           unsigned long long int    Number of vnodes extensions
                                                   that were cleaned
     reserved2           hyper[11]                 Reserved
  EFS_STATS_API_STRUCT2
     reserved            unsigned long long int    Reserved
     grand_total_vnodes  unsigned long long int    Total count of vnode ops
     total_ops           unsigned long long int    Number of vnode op counts
     convert_namecount   unsigned long long int    Count of names processed
                                                   during auto conversion for
                                                   version 2, reserved for
                                                   version 1.
     v2dir_splits        unsigned long long int    V5 directory bucket splits
     v2dir_merges        unsigned long long int    V5 directory bucket merges
     reserved1           hyper[9]                  Reserved
     _Packed ZFSVNODEOPCOUNTS[50]
     opname              char[26]                  Vnode operation name
     pad1                char[2]                   Reserved
     opcount             unsigned long long int    Count of vnode op requests
     reserved            hyper[2]                  Reserved
  reserved               hyper[10]                 Reserved
systemname               char[9]                   Name of system to get stats
```

```
Return_value     0 if request is successful, -1 if it is not successful

Return_code
  EINTR          zFS is shutting down
  EINVAL         Invalid parameter list
  EMVSERR        Internal error occurred
  E2BIG          Information too big for buffer supplied

Reason_code
  0xEFnnxxxx     See z/OS Distributed File Service Messages and Codes
```

Usage notes

- Reserved fields and undefined flags must be set to binary zeros.
- Version 1 provided 8-byte counters but only used the low order 4-bytes. Version 2 uses full 8-byte counters.
- Same named fields in version 1 and 2 that are not reserved start at the same offset.

Privilege required

None.

Related services

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 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 int        high; /* unsigned int reserved */
    unsigned int        low;
} hyper;

/* reset timestamp */
typedef struct reset_time {
    unsigned int        posix_time_high; /* high order 32 bits since epoc */
    unsigned int        posix_time_low;  /* low order 32 bits since epoch */
    unsigned int        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; /* decimal part is in thousandths */
    /* 3 means .003 and 300 means .3 */
    hyper               Allocates;
    hyper               Deletes;
    hyper               VnodeStructSize;
    hyper               ExtendedVnodes;
    hyper               extensionSize; /* (minimum) in bytes */
    hyper               USSHeldVnodes;
    hyper               USSHeldVnodesHi;
    hyper               OpenVnodes;
    hyper               OpenVnodesHi;
    hyper               OpenVnodesReuse;
    int                 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;
    int                 convert_namecount;
    int                 reserved1[3];
    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;

typedef struct VNM_STATS_API_STRUCT2_T
{
    unsigned long long int reserved;
    unsigned long long int Vnodes;
    unsigned long long int Requests;
    unsigned long long int Hits;
    hyper                  RatioWhole;
    hyper                  RatioDecimal; /* decimal part is in thousandths */
```

```
        /* 3 means .003 and 300 means .3 */
        unsigned long long int Allocates;
        unsigned long long int Deletes;
        unsigned long long int VnodeStructSize;
        unsigned long long int ExtendedVnodes;
        unsigned long long int extensionSize; /* (minimum) in bytes */
        unsigned long long int USSHeldVnodes;
        unsigned long long int USSHeldVnodesHi;
        unsigned long long int OpenVnodes;
        unsigned long long int OpenVnodesHi;
        unsigned long long int OpenVnodesReuse;
        unsigned long long int extCleans;
        int                    reserved1[2];
        hyper                  reserved2[10];
    } VNM_STATS_API_STRUCT2;

    typedef _Packed struct zFSVnodeOpCounts_t {
        char                   opname[26];   /* Operation being counted */
        char                   pad1[2];
        unsigned long long int opcount;      /* Number of operations performed */
        hyper                  reserved[2];  /* reserved for future use */
    } _Packed zFSVnodeOpCounts;

    typedef struct EFS_STATS_API_STRUCT2_T
    {
        unsigned long long int reserved;
        unsigned long long int grand_total_vnodes;
        unsigned long long int total_ops;
        unsigned long long int convert_namecount;
        unsigned long long int v5dir_splits;
        unsigned long long int v5dir_merges;
        hyper                  reserved2[9];
        _Packed zFSVnodeOpCounts zFSOpCounts[50];
    } EFS_STATS_API_STRUCT2;

    typedef struct stat_vnode_cache2_t
    {
        VNM_STATS_API_STRUCT2 vnm_stats_info;
        EFS_STATS_API_STRUCT2 efs_stats_info;
        hyper                reserved[10];
    } STAT_VNODE_CACHE2;

    /*********************************************************************/
    /* 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
    #define                  SA_VER_2       0x02
        char                 sa_flags;  /* flags field, x80 means reset stats    */
    #define                  SA_RESET 0x80
        char                 sa_fill[3];    /* spare bytes */
        int                  sa_supported_ver; /* version of data returned */
        int                  sa_reserve[3]; /* Reserved */
        struct reset_time    reset_time_info;
    } STAT_API;

    struct parmstruct {
        syscall_parmlist  myparms;
        STAT_API          myapi;
        STAT_VNODE_CACHE2 mystats;
        char              systemname[9];
    } myparmstruct;

    int main(int argc, char **argv)
    {
        int      bpxrv;
        int      bpxrc;
        int      bpxrs;
        int      i;
        double   temp_ratio;
        int      whole;
        int      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;
        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_2;
        stapptr->sa_len = (int)sizeof(STAT_VNODE_CACHE2);

        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;
        }
        else
        {
          if (stapptr->sa_supported_ver == SA_VER_INITIAL)
          {
            /* Print the version 1 ouput */
            STAT_VNODE_CACHE *mystatsp = (STAT_VNODE_CACHE *)&myparmstruct.mystats;
            i = 0;
            printf("%50s\n", "zFS Vnode Op Counts");
            printf(" \n");
            printf("Vnode Op                  Count       "
                   "Vnode Op                  Count \n");
            printf("------------------------  ----------  "
                   "------------------------  ---------- \n");

            while (i < mystatsp->efs_stats_info.total_ops.low)
            {
                printf("%-25s %10u  ",
                       mystatsp->efs_stats_info.zFSOpCounts[i].opname,
                       mystatsp->efs_stats_info.zFSOpCounts[i++].opcount.low);
                if (i < mystatsp->efs_stats_info.total_ops.low)
                {
                    printf("%-25s %10u\n",
                           mystatsp->efs_stats_info.zFSOpCounts[i].opname,
                           mystatsp->efs_stats_info.zFSOpCounts[i++].opcount.low);
                }
            }
            printf("\nTotal zFS Vnode Ops       %10u\n\n",
                   mystatsp->efs_stats_info.grand_total_vnodes.low);
            printf("%52s\n", "zFS Vnode Cache Statistics");
            printf(" \n");
            printf(" Vnodes     Requests   Hits       Ratio   "
                   "Allocates  Deletes\n");
            printf(" ---------- ---------- ---------- ------- "
                   "---------- ----------\n");
            printf("%10u %10u %10u %3u.%1.1u%% %10u %10u\n",
                   mystatsp->vnm_stats_info.Vnodes.low,
                   mystatsp->vnm_stats_info.Requests.low,
                   mystatsp->vnm_stats_info.Hits.low,
                   mystatsp->vnm_stats_info.RatioWhole.low,
                   mystatsp->vnm_stats_info.RatioDecimal.low,
                   mystatsp->vnm_stats_info.Allocates.low,
                   mystatsp->vnm_stats_info.Deletes.low);

            printf(" \n");
            printf("zFS Vnode structure size: %u bytes\n",
                   mystatsp->vnm_stats_info.VnodeStructSize.low);

            printf("zFS extended vnodes: %u, extension size %u bytes (minimum)\n",
                   mystatsp->vnm_stats_info.ExtendedVnodes.low,
                   mystatsp->vnm_stats_info.extensionSize.low);

            printf("Held zFS vnodes: %10u (high %10u) \nOpen zFS vnodes: %10u "
                   "(high %10u) Reusable: %u\n",
                   mystatsp->vnm_stats_info.USSHeldVnodes.low,
                   mystatsp->vnm_stats_info.USSHeldVnodesHi.low,
                   mystatsp->vnm_stats_info.OpenVnodes.low,
                   mystatsp->vnm_stats_info.OpenVnodesHi.low,
                   mystatsp->vnm_stats_info.OpenVnodesReuse.low);
            printf(" \n");

            if (0 == ctime_r((time_t * )&stapptr->reset_time_info.posix_time_low,
                             buf))
                printf("Could not get timestamp.\n");
            else
            {   /* Insert the microseconds into the displayable time value */
                strncpy(&(buf[27]), &(buf[20]), 6);
                sprintf(&(buf[20]), "%06d", stapptr->reset_time_info.posix_usecs);
                buf[26] = ' ';
                buf[19] = '.';
                printf("Last Reset Time: %s", buf);
            }
          }
          else
          {
            /* Print the version 2 ouput */
            STAT_VNODE_CACHE2 *mystatsp = &myparmstruct.mystats;
            i = 0;
            printf("%50s\n", "zFS Vnode Op Counts");
            printf(" \n");
            printf("Vnode Op                  Count       "
                   "Vnode Op                  Count \n");
            printf("------------------------  ----------  "
                   "------------------------  ---------- \n");
            while (i < mystatsp->efs_stats_info.total_ops)
            {
                printf("%-25s %10llu  ",
                       mystatsp->efs_stats_info.zFSOpCounts[i].opname,
                       mystatsp->efs_stats_info.zFSOpCounts[i++].opcount);
                if (i < mystatsp->efs_stats_info.total_ops)
                {
                    printf("%-25s %10llu\n",
                           mystatsp->efs_stats_info.zFSOpCounts[i].opname,
                           mystatsp->efs_stats_info.zFSOpCounts[i++].opcount);
                }
            }
            printf("\nTotal zFS Vnode Ops       %10llu\n\n",
                   mystatsp->efs_stats_info.grand_total_vnodes);
            printf("%52s\n", "zFS Vnode Cache Statistics");
            printf(" \n");
            printf(" Vnodes     Requests   Hits       Ratio   "
                   "Allocates  Deletes\n");
            printf(" ---------- ---------- ---------- ------- "
                   "---------- ----------\n");
            printf("%10llu %10llu %10llu %3llu.%1.1llu%% %10llu %10llu\n",
                   mystatsp->vnm_stats_info.Vnodes,
                   mystatsp->vnm_stats_info.Requests,
                   mystatsp->vnm_stats_info.Hits,
                   mystatsp->vnm_stats_info.RatioWhole,
                   mystatsp->vnm_stats_info.RatioDecimal,
                   mystatsp->vnm_stats_info.Allocates,
                   mystatsp->vnm_stats_info.Deletes);

            printf(" \n");
            printf("zFS Vnode structure size: %llu bytes\n",
                   mystatsp->vnm_stats_info.VnodeStructSize);

            printf("zFS extended vnodes: %llu, extension size %llu "
                   "bytes (minimum)\n",
                   mystatsp->vnm_stats_info.ExtendedVnodes,
                   mystatsp->vnm_stats_info.extensionSize);

            printf("Held zFS vnodes: %10llu (high %10llu) \nOpen zFS vnodes: "
                   "%10llu (high %10llu) Reusable: %llu\n",
                   mystatsp->vnm_stats_info.USSHeldVnodes,
                   mystatsp->vnm_stats_info.USSHeldVnodesHi,
                   mystatsp->vnm_stats_info.OpenVnodes,
                   mystatsp->vnm_stats_info.OpenVnodesHi,
                   mystatsp->vnm_stats_info.OpenVnodesReuse);
            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;
    }
```

Unquiesce Aggregate

Purpose

An aggregate operation that unquiesces a zFS compatibility mode aggregate on a system. This subcommand call allows activity on the aggregate and its file system to resume.

Format

```
syscall_parmlist
   opcode          int                 133       AGOP_UNQUIESCE_PARMDATA
   parms[0]        int                 offset to AGGR_ID
   parms[1]        int                 quiesce handle
   parms[2]        int                 0
   parms[3]        int                 0
   parms[4]        int                 0
   parms[5]        int                 0
   parms[6]        int                 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
  EMVSERR       Internal error using an osi service
  ENOENT        Aggregate is not attached
  EPERM         Permission denied to perform request

Reason_code
  0xEFnnxxxx    See z/OS Distributed File Service Messages and Codes
```

Usage notes

- The unquiesce call must supply the quiesce handle that was returned by the quiesce call. The aggregate is typically quiesced before backing up the aggregate. After the backup is complete, the aggregate can be unquiesced.
- Reserved fields and undefined flags must be set to binary zeros.

Privilege required

The issuer must be logged in as root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL resource in the z/OS UNIXPRIV class.

Related services

Quiesce Aggregate

Restrictions

None.

Examples

```
#pragma linkage(BPX1PCT, OS)
#pragma LANGLVL(EXTENDED)

extern void BPX1PCT(char *, int, int, char *, int *, int *, int *);

#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_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] = "PLEX.DCEIMGQX.FS";
  int              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;

  /* Ensure reserved fields are 0 */
  memset(&myparmstruct.aggr_id, 0, sizeof(AGGR_ID));
  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 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;
  }
```

Part 3. Appendixes

Appendix A. Running the zFS pfsctl APIs in 64-bit mode

The pfsctl (BPX1PCT) application programming interface can be invoked in a 64-bit environment. To do this, you must take the following steps:

1. Replace the BPX1PCT with BPX4PCT
2. Replace the `#pragma linkage(BPX1PCT, OS)` statement with `#pragma linkage(BPX4PCT, OS64_NOSTACK)`
3. Ensure that there are appropriate includes for function calls
4. Ensure all functions that require 64-bit parameters are passing 64-bit numbers (for example, ctime_r).

The remaining code is, or can remain, unchanged. "Statistics iocounts information (64-bit mode)" shows example code that were updated to be invoked in a 64-bit environment.

Statistics iocounts information (64-bit mode)

Examples

```
#pragma linkage(BPX4PCT, OS64_NOSTACK)
extern void BPX4PCT(char *, int, int, char *, int *, int *, int *);

#include <stdio.h>
#include <time.h>

#define ZFSCALL_STATS    0x40000007
#define STATOP_IOCOUNTS                  243  /* Performance API queries */

#define TOTAL_TYPES       3
#define TOTAL_CIRC       19

#define u_int unsigned int

typedef struct syscall_parmlist_t
{
    int opcode;              /* Operation code to perform            */
    int parms[7];            /* Specific to type of operation,       */
                             /* provides access to the parms         */
                             /* parms[4]-parms[6] are currently unused*/
} syscall_parmlist;

typedef  struct  reset_time {
  u_int   posix_time_high;   /* high order 32 bits since epoc */
  u_int   posix_time_low;    /* low order 32 bits since epoch */
  u_int   posix_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 API_IO_BY_TYPE_t
{
    unsigned int number_of_lines;
    unsigned int count;
    unsigned int waits;
    unsigned int cancels;  /* Successful cancels of IO */
    unsigned int merges;   /* Successful merges of IO */
    char     reserved1[6];
    char     description[51];
    char     pad1[3];
} API_IO_BY_TYPE;

typedef struct API_IO_BY_CIRC_t
{
    unsigned int number_of_lines;
    unsigned int count;
    unsigned int waits;
    unsigned int cancels;
    unsigned int merges;
    char     reserved1[6];
    char     description[51];
    char     pad1[3];
} API_IO_BY_CIRC;

/*********************************************************************/
/* The following structures are used to represent cfgop queries      */
/* for iocounts                                                      */
/*********************************************************************/

struct parmstruct
{
    syscall_parmlist myparms;
    STAT_API myapi;
    API_IO_BY_TYPE mystatsbytype[TOTAL_TYPES];
    API_IO_BY_CIRC mystatsbycirc[TOTAL_CIRC];
} myparmstruct;

int main(int argc, char **argv)
{
    int bpxrv;
    int bpxrc;
    int bpxrs;
    int i;

    STAT_API *stapptr = &(myparmstruct.myapi);
    API_IO_BY_TYPE *stiotptr = &(myparmstruct.mystatsbytype[0]);
    API_IO_BY_CIRC *stiocptr = &(myparmstruct.mystatsbycirc[0]);

    char buf[33];

    myparmstruct.myparms.opcode = STATOP_IOCOUNTS;
    myparmstruct.myparms.parms[0] = sizeof(syscall_parmlist);
    myparmstruct.myparms.parms[1] = sizeof(syscall_parmlist) + sizeof(STAT_API);
    myparmstruct.myparms.parms[2] = 0;
    myparmstruct.myparms.parms[3] = 0;
    myparmstruct.myparms.parms[4] = 0;
    myparmstruct.myparms.parms[5] = 0;
    myparmstruct.myparms.parms[6] = 0;

    memset(stapptr,0,sizeof(STAT_API));
    memcpy(stapptr->sa_eye,SA_EYE,4);
    stapptr->sa_ver=SA_VER_INITIAL;
    stapptr->sa_len=(int) (TOTAL_TYPES * sizeof(API_IO_BY_TYPE))
                        + (TOTAL_CIRC * sizeof(API_IO_BY_CIRC));

    BPX4PCT("ZFS     ",
            ZFSCALL_STATS,            /* Perf statistics operation        */
            sizeof(myparmstruct),     /* Length of Argument      */
            (char *) &myparmstruct,   /* Pointer to Argument     */
            &bpxrv,                   /* Pointer to Return_value */
            &bpxrc,                   /* Pointer to Return_code  */
            &bpxrs);                  /* Pointer to Reason_code  */
    if( bpxrv < 0 )
    {
       printf("Error querying iocounts, BPXRV = %d BPXRC = %d BPXRS = %x\n",bpxrv,bpxrc,bpxrs);
       return bpxrc;
    }
    else
    {
        if( stiotptr->number_of_lines != TOTAL_TYPES )
        {
            printf("Unexpected number of IO Types, %d instead of TOTAL_TYPES\n",
            stiotptr->number_of_lines);
            return 1;
        }
        if( stiocptr->number_of_lines != TOTAL_CIRC )
        {
            printf("Unexpected number of IO Circumstances, %d instead of TOTAL_CIRC\n",
            stiocptr->number_of_lines);
            return 2;
        }
        printf("                  I/O Summary By Type\n");
        printf("                  -------------------\n");
        printf("\n");
     printf("Count      Waits      Cancels    Merges     Type\n");
     printf("---------- ---------- ---------- ---------- ----------\n");
     for( i=0; i<TOTAL_TYPES; i++ )
     {
         printf("%10u  %10u  %10u  %10u  %s\n",
                stiotptr->count, stiotptr->waits,
                stiotptr->cancels, stiotptr->merges,
                stiotptr->description);
         stiotptr = stiotptr + 1;
     }
     printf("\n");
     printf("                  I/O Summary By Circumstance\n");
     printf("                  ---------------------------\n");
     printf("\n");
     printf("Count      Waits      Cancels    Merges     Circumstance\n");
     printf("---------- ---------- ---------- ---------- ------------\n");
     for( i=0; i<TOTAL_CIRC; i++ )
     {
         printf("%10u  %10u  %10u  %10u  %s\n",
                   stiocptr->count, stiocptr->waits,
                   stiocptr->cancels, stiocptr->merges,
                   stiocptr->description);
            stiocptr = stiocptr +1;
            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",buf);
        }
    }
    return 0;
}
```

Statistics iocounts information (64-bit) mode

Appendix B. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS V2R2 ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, `3` or `3.1` or `3.1.1`. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines `3.1 USERID` and `3.1 SYSTEMID`, your syntax can include either `USERID` or `SYSTEMID`, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The `*` symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element `*FILE` with dotted decimal number 3 is given the format `3 \* FILE`. Format `3* FILE` indicates that syntax element `FILE` repeats. Format `3* \* FILE` indicates that syntax element `* FILE` repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines `5.1*, 5.1 LASTRUN, and 5.1 DELETE` mean that if you use more than one of the `LASTRUN` and `DELETE` syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the `%` symbol, it indicates a reference that is defined elsewhere. The string that follows the `%` symbol is the name of a syntax fragment rather than a literal. For example, the line `2.1 %OP1` means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

`? indicates an optional syntax element`
: The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example `5? NOTIFY`). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines `5 ?, 5 NOTIFY`, and `5 UPDATE`, you know that the syntax elements `NOTIFY` and `UPDATE` are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

`! indicates a default syntax element`
: The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines `2? FILE, 2.1! (KEEP), and 2.1 (DELETE)`, you know that `(KEEP)` is the

default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

`* indicates an optional syntax element that is repeatable`
The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

`+ indicates a syntax element that must be included`
The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming Interface Information

This publication, *z/OS Distributed File Service zFS Administration*, primarily documents information that is NOT intended to be used as Programming Interfaces of the Distributed File Service.

z/OS Distributed File Service zFS Administration also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of the Distributed File Service. This information is identified where it occurs by an introductory statement to a chapter or section or by the following marking.

```
[--- NOT Programming Interface information ---]
[--- End of NOT Programming Interface information ---]
```

Trademarks

Glossary

This glossary includes terms and definitions for Distributed File Service z/OS File System. The following cross-references are used in this glossary:

1. See refers the reader from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
2. See also refers the reader to a related or contrasting term.

To view glossaries for other IBM products, go to www.ibm.com/software/globalization/terminology.

aggregate
A structured collection of data objects that form a data type.

attach In z/OS, to create a task that can execute concurrently with the attaching code.

audit identifier
In zFS, a 16-byte value associated with each z/OS UNIX file or directory that provides identity in an SMF audit record or in certain authorization failure messages.

bitmap
In zFS, a file listing the blocks that are free on disk. The file size is dependent on the size of the aggregate.

catch-up mount
A local mount that z/OS UNIX automatically issues to every other system's physical file system that is running sysplex-aware for that mode (read-write or read-only) when a sysplex-aware file system mount is successful on a system in a shared file system environment.

compatibility mode aggregate
A Virtual Storage Access Method linear data set (VSAM LDS) that contains a single read-write zFS file system.

DFS See Distributed File Service.

Distributed File Service (DFS)
A base element of z/OS that allows users to access and share data in a distributed environment across a wide range of IBM and non-IBM platforms.

EAV See extended address volume.

extended address volume (EAV)
DASD storage that can contain more than 65,521 cylinders per volume.

file handle
A number that is used by the client and server sides of the Network File System (NFS) or the Server Message Block (SMB) to specify a particular file or prefix.

file system owner
In z/OS, the system that coordinates sysplex activity for a particular file system.

function shipping
The process of requesting function from to the owning file system and returning the response to the requester through XCF communications.

global resource serialization
A component of z/OS that serializes the use of system resources and converts hardware reserves on direct access storage device (DASD) volumes to data set enqueues.

global resource serialization complex
A group of systems that use global resource serialization to serialize access to shared resources such as data sets on shared direct access storage device (DASD) volumes.

hang To become unresponsive to user commands and to stop or appear to stop processing.

i-node The internal structure that describes the individual files in the UNIX file system. An i-node contains the node, type, owner, and location of a file.

local mount
A mount that is known to the physical file system.

metadata
Data that describes the characteristics of data; descriptive data.

non-sysplex aware
A mounted file system that has file

requests handled by remotely function shipping requests through z/OS UNIX

root file system
The basic file system onto which all other file systems can be mounted. The root file system contains the operating system files that run the rest of the system.

salvager
In zFS, a program that examines a zFS aggregate to determine if there are any inconsistencies in the structure of the aggregate.

Server Message Block (SMB)
A protocol that manages requests and responses in a client/server environment so that clients on a network can share files, directories, and devices.

SMB See Server Message Block.

sysplex
A set of z/OS systems that communicate with each other through certain multisystem hardware components and software services.

sysplex-aware
A mounted file system that has file requests handled locally instead of function shipping requests through z/OS UNIX.

version file system
See root file system.

zFS See z/OS file system.

zFS aggregate
A Virtual Storage Access Method Linear Data Set (VSAM LDS) that contains a zFS file system.

z/OS File System (zFS)
A type of file system that resides in a Virtual Storage Access Method (VSAM) linear data set (LDS) and has a hierarchical organization of files and directories with a root directory.

Index

Product Number: 5650-ZOS

Printed in USA

SC23-6887-01